DATA TRANSLATION

UM-19593-Q

DT9840 Series Host Communication Library User's Manual

Fifteenth Edition
August, 2013

Copyright © 2013 by Data Translation, Inc.
All rights reserved.

Information furnished by Data Translation, Inc. is believed to be accurate and reliable; however, no responsibility is assumed by Data Translation, Inc. for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent rights of Data Translation, Inc.

Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer software clause at 48 C.F.R, 252.227-7013, or in subparagraph (c)(2) of the Commercial Computer Software - Registered Rights clause at 48 C.F.R., 52-227-19 as applicable. Data Translation, Inc., 100 Locke Drive, Marlboro, MA 01752.

Data Translation® is a registered trademark of Data Translation, Inc.

All other brand and product names are trademarks or registered trademarks of their respective companies.

Data Translation, Inc.
100 Locke Drive
Marlboro, MA 01752-1192
(508) 481-3700
www.datatranslation.com
Fax: (508) 481-8620
E-mail: info@datx.com

Table of Contents

About this Manual 7
Intended Audience 7
What You Should Learn from this Manual 7
Conventions Used in this Manual 8
Related Documents 8
Where to Get Help 8

Chapter 1: Overview 9
Introduction 10
Board-Level Functions 11
Channel-Level Functions 12
Conversion Functions 13
Installing the Software 15
Building Host Applications Using the DT9840 Series Host Communication Library 16
Product Support 17
General Checklist 17
Service and Support 17

Chapter 2: Using the DT9840 Series Host Communication Library 19
Overview 20
Opening and Closing a Module 23
Downloading and Running a DSP Program 25
Opening and Closing a Communication Channel 26
Reading Data from a Communication Channel 27
Writing Data to a Communication Channel 29
Handling Channel Communication Events 31
Converting Data 32
Handling Errors 34

Chapter 3: Programming Flowcharts 35
Typical Communication Operations 37
Typical Event-Driven Communication Operations 39

Chapter 4: Function Reference for Visual C++ Programmers 43
DT_ADBlockToVolts 45
DT_ADChanBlockToVolts 46
DT_ADScanToVolts 47
DT_ADValueToVolts 48
DT_BoardClose 49
DT_BoardDownload 50

DT_BoardGetInfo 51
DT_BoardGetName 51
DT_BoardOpen 52
DT_BoardReadFromMemory 54
DT_BoardReadRegister 55
DT_BoardRun 56
DT_BoardWriteRegister 57
DT_BoardWriteToMemory 58
DT_ChanAcknowledge 59
DT_ChanAsyncWrite 60
DT_ChanBytesAvailable 61
DT_ChanClose 61
DT_ChanGetInfo 62
DT_ChanOpen 63
DT_ChanRead 65
DT_ChanReadAvailable 67
DT_ChanSyncWrite 68
DT_GetErrorString 69
DT_RegisterCallback 70
DT_RegisterMsgHandler 73
DT_UnregisterCallback 75
DT_UnregisterMsgHandler 76
DT_VoltsToDABlock 77
DT_VoltsToDAChanBlock 78
DT_VoltsToDAScan 80
DT_VoltsToDAValue 81

Chapter 5: Function Reference for Visual Basic Programmers 83

DT_ADScanToVolts 84
DT_ADValueToVolts 85
DT_BoardClose 85
DT_BoardDownload 86
DT_BoardGetInfo 87
DT_BoardGetName 88
DT_BoardOpen 89
DT_BoardReadFromMemory 90
DT_BoardReadRegister 91
DT_BoardRun 92
DT_BoardWriteRegister 93
DT_BoardWriteToMemory 94
DT_ChanAsyncWrite 95
DT_ChanBytesAvailable 96
DT_ChanClose 97
DT_ChanGetInfo 98
DT_ChanOpen 99
DT_ChanRead 101
DT_ChanReadAvailable 103

DT_GetErrorString 104
DT_VoltsToDAScan 105
DT_VoltsToDAValue 106

Appendix A: Error Codes 107

Appendix B: Data Types, Constants, Enumerated Types, Structures, and Callback Functions 115

For Visual C++ Programmers 116
Data Types 116
Constants 116
General-Purpose Constants 117
Conversion Constants 118
Register Constants 119
Error Constants 121
Event Constants 122
Message Constants 122
Counter/Timer Constants 122
Flash Memory Constants 123
Enumerated Types 123
Structures 139
DATA_FILE_HDR or *PDATA_FILE_HDR Structure 151
Callback Functions 162
For Visual Basic Programmers 164
Constants 164
Global Constants 164
Conversion Constants 165
Register Constants 166
Error Constants 169
Counter/Timer Constants 169
Flash Memory Constants 169
Enumerated Types 169
Data Types 186

Appendix C: Communication Events and Messages 207

DTEVENT_AD_OVERRUN 208
DTEVENT_ASYNC_DATA_RECEIVED 208
DTEVENT_CHAN_IN_BUFFER_FULL 209
DTEVENT_DA_UNDERRUN 210
DTEVENT_ERROR 210
DTEVENT_INPUT_SCAN_BLOCK_OVERRUN 211
DTEVENT_MODULE_PLUGGED 211
DTEVENT_MODULE_UNPLUGGED 212
DTEVENT_OUT_BUFFER_SPACE_AVAILABLE 213
DTEVENT_OUTPUT_SCAN_BLOCK_UNDERRUN 214

DTEVENT_REMOTE_CHAN_CLOSED 214
DTEVENT_REMOTE_CHAN_OPENED 215
DTEVENT_SYNC_DATA_RECEIVED 215

Index 217

About this Manual

This manual describes how to use the functions included in the DT9840 Series Host Communication Library to create a Windows-based application that runs on the host computer and communicates with a DSP program running on a DT9840 Series module.

Intended Audience

This document is intended for Visual C++ or Visual Basic programmers who are responsible for writing a host application program that communicates with the DSP on a DT9840 Series module. The DT9840 Series Host Communication Library is supported in Microsoft®Windows® XP, Windows Vista®, or Windows 7 using Microsoft Visual Studio 6.0 or greater.

What You Should Learn from this Manual

This manual provides detailed information about the functions and messages provided in the DT9840 Series Host Communication Library. This manual is organized as follows:

- Chapter 1, "Overview," provides an introduction to the DT9840 Series Host Communication Library.
- Chapter 2, "Using the DT9840 Series Host Communication Library," provides conceptual information to help you use the functions in the DT9840 Series Host Communication Library.
- Chapter 3, "Programming Flowcharts," provides flow diagrams that show how to use the functions together to write a host application program that communicates with a DSP program on a DT9840 Series module.
- Chapter 4, "Function Reference for Visual C++ Programmers," describes each of the host communication functions in detail for Visual C++ programmers.
- Chapter 5, "Function Reference for Visual Basic Programmers," describes each of the host communication functions in detail for Visual C++ programmers.
- Appendix A, "Error Codes," lists the errors that can be returned or detected by the DT9840 Series Host Communication Library.
- Appendix B, "Data Types, Constants, Enumerated Types, Structures, and Callback Functions," describes the data types, constants, enumerated types, structures, and callback functions that are used by the DT9840 Series Host Communication Library.
- Appendix C, "Communication Events and Messages," describes the messages that can be returned when communication events occur in the system.
- An index completes this manual.

Conventions Used in this Manual

The following conventions are used in this manual:

- Notes provide useful information or information that requires special emphasis, cautions provide information to help you avoid losing data or damaging your equipment, and warnings provide information to help you avoid catastrophic damage to yourself or your equipment.
- Items that you select or type are shown in **bold**.

Related Documents

Refer to the following documents for more information:

- *DT9840 Series Getting Started Manual* (UM-19199). This manual, included on the DT9840 Series Software CD, describes the how to install a DT9840 Series module and related software.
- *DT9840 Series User's Manual* (UM-19197). This manual, included on the DT9840 Series Software CD, describes the hardware features of the DT9840 Series function modules, including the hardware registers, specifications, and connector pin assignment information.
- *DT9840 Series DSP Library User's Manual* (UM-19591). This manual, included on the DT9840 Series Software CD, describes how to write a DSP program to perform communication, I/O, and conversion operations on a DT9840 Series module.
- Documentation for Code Composer Studio™ Integrated Development Environment (IDE) from Texas Instruments.
- Documentation for Texas Instruments TMS320C6713 DSP processor.
- Microsoft Windows XP, Windows Vista, or Windows 7 documentation.
- Microsoft Visual Studio documentation.
- USB 1.1 and USB 2.0 specifications available on the USB web site (http://www.usb.org).

Where to Get Help

Should you run into problems installing or using the DT9840 Series Host Communication Library, the Data Translation Technical Support Department is available to provide technical assistance. Refer to page 17 for more information. If you are outside the United States or Canada, call your local distributor, whose number is listed on Data Translation's web site (www.datatranslation.com).

Overview

Introduction 10
Installing the Software 15
Building Host Applications Using the DT9840 Series Host Communication Library 16
Product Support 17

Introduction

The DT9840 Series Host Communication Library allows the host computer to communicate with and control a DSP application on a DT9840 Series module. The software architecture is shown in Figure 1.

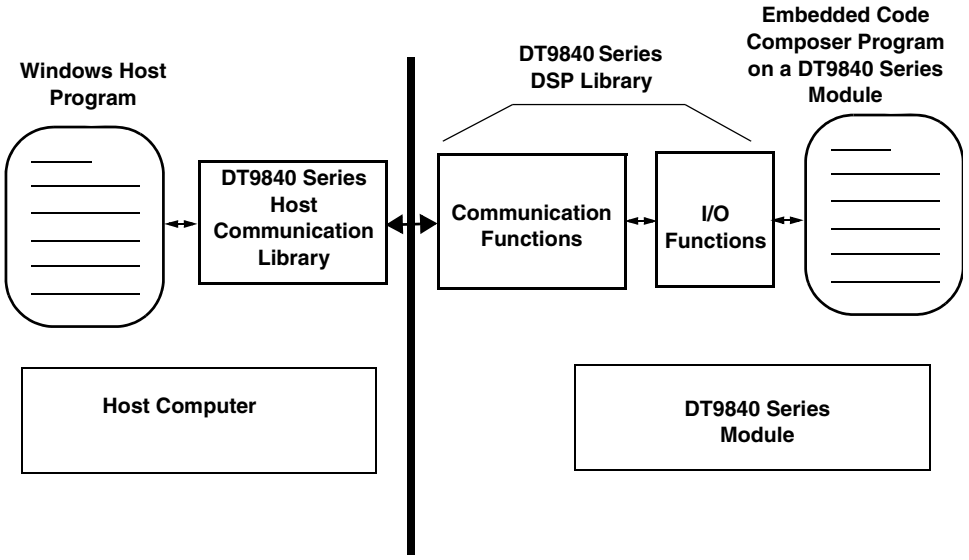


Figure 1: DT9840 Series Software Architecture

To communicate between the host computer and a DT9840 Series module, you need to write an application program for the host computer that calls the communication functions in the DT9840 Series Host Communication Library. In addition, you need to write a Code Composer program for the DSP on the DT9840 Series module that calls the matching communication functions in the DT9840 Series DSP Library. Refer to the *DT9840 Series DSP Library User's Manual* for more information about writing a DSP program for a DT9840 Series module.

The DT9840 Series Host Communication Library simplifies the details of the USB communication channel that allows a host application program to communicate with a DSP program on a DT9840 Series module. The communication model is stream-based. The DT9840 Series Host Communication Library provides three types of API functions:

- Board-level functions, described on this page
- Channel-level functions, described on page 12
- Conversion functions, described on page 13

Board-Level Functions

Board-level communication functions allow you to open and a close a DT9840 Series module, get information about a module, get information about error/status codes that are returned, download and run programs on a module, read data from and write to DSP memory on a module, and handle communication events.

Table 1 summarizes the board-level communication functions. To better understand how to use these functions, refer to Chapter 2 starting on page 19. For detailed syntax information in Visual C++, refer to Chapter 4 starting on page 43. For detailed syntax information in Visual Basic, refer to Chapter 5 starting on page 83.

Table 1: Board-Level Functions

Function Type	Function Name	Description
Open and Close Functions	DT_BoardOpen	Opens a DT9840 Series module by name, specifies the size of the internal messaging buffers for the module, and returns a handle to the module.
	DT_BoardClose	Closes a previously opened module and releases all allocated resources that were associated with it.
Download and Run Functions	DT_BoardDownload	Downloads a compiled DSP program over the USB bus to the DSP on a DT9840 Series module.
	DT_BoardRun	Runs a previously downloaded DSP program on a DT9840 Series module.
Write Functions	DT_BoardWriteToMemory	Writes data from a variable on the host computer to an address in DSP memory on a DT9840 Series module (with no assistance from the DSP).
	DT_BoardWriteRegister	Writes a value to a specified register on a DT9840 Series module.
Read Functions	DT_BoardReadFromMemory	Reads data from an address in DSP memory on a DT9840 Series module to a variable on the host computer (with no assistance from the DSP).
	DT_BoardReadRegister	Reads the value of a specified register on the DT9840 Series module.
Communication Support Functions	DT_BoardGetName	Returns the name of a connected DT9840 Series module.
	DT_BoardGetInfo	Returns information about the type of module that is connected.
	DT_GetErrorString	Returns an ASCII string that corresponds to specified error code.

Table 1: Board-Level Functions (cont.)

Function Type	Function Name	Description
Event Handling Functions	DT_RegisterMsgHandler[a]	Registers the window in which all future Windows messages that are associated with specified events are received.
	DT_UnregisterMsgHandler[a]	Unregisters a previously registered Windows message handler.
	DT_RegisterCallback[a]	Registers a callback function that is called whenever specified events occur.
	DT_UnregisterCallback[a]	Unregisters a previously registered callback function.

a. This function is not supported in Visual Basic.

Channel-Level Functions

Channel-level functions perform various operations on a single communication channel, which connects a program running on the host computer with a DSP program running on a DT9840 Series module. Operations include opening and closing a channel, getting information about a channel, reading data from a channel, and writing data to a channel.

Table 2 summarizes the channel-level communication functions. To better understand how to use these functions, refer to Chapter 2 starting on page 19. For detailed syntax information in Visual C++, refer to Chapter 4 starting on page 43. For detailed syntax information in Visual Basic, refer to Chapter 5 starting on page 83.

Table 2: Channel-Level Communication Functions

Function Type	Function Name	Description
Channel Open and Close Functions	DT_ChanOpen	Opens a unidirectional stream-based communication channel by name, specifies the size of the channel input buffer if the channel was opened for reading, and returns a handle to the channel.
	DT_ChanClose	Closes a previously opened communication channel on the host and releases all allocated resources that were associated with it.
Channel Write Functions	DT_ChanAsyncWrite	Asynchronously writes a specified number of bytes from an open communication channel on the host to a DT9840 Series module.
Channel Write Functions (cont.)	DT_ChanSyncWrite[a]	Synchronously writes a specified number of bytes from an open communication channel on the host to a DT9840 Series module and waits for a response from the module.
	DT_ChanAcknowledge[a]	Acknowledges that the host received a synchronous write command from a DT9840 Series module.

Table 2: Channel-Level Communication Functions (cont.)

Function Type	Function Name	Description
Channel Read Functions	DT_ChanRead	Reads a specified number of bytes from a previously opened communication channel.
	DT_ChanReadAvailable	Returns the data that is currently available from an open communication channel on the DT9840 Series module, up to the size of the channel input buffer.
Channel Communication Support Functions	DT_ChanGetInfo	Returns information about a specified channel in a list of channels.
	DT_ChanBytesAvailable	Returns the number of bytes that are currently available in the specified channel buffer.

a. This function is not supported in Visual Basic.

Conversion Functions

Conversion functions allow you to convert raw analog data into voltage values and to convert voltage values into raw analog data.

Table 3 summarizes the conversion functions. To better understand how to use these functions, refer to Chapter 2 starting on page 19. For detailed syntax information in Visual C++, refer to Chapter 4 starting on page 43. For detailed syntax information in Visual Basic, refer to Chapter 5 starting on page 83.

Table 3: Conversion Functions

Function Type	Function Name	Description[a]
Conversion to Voltage Functions	DT_ADValueToVolts	Converts a single raw analog input value that is returned by **DT_ReadADC** into a voltage value.
	DT_ADScanToVolts	Converts a scan of analog input values that is returned by **DT_ReadADCDIO**, **DT_CaptureBuffer**, or **DT_ScanLoop** into an array of voltage values.
	DT_ADChanBlockToVolts[b]	For a specified analog input channel in a block, converts the raw analog values returned by **DT_BlockLoop** or **DT_ListLoop** into an array of voltage values.
	DT_ADBlockToVolts[b]	For all the analog input channels in a block, converts the raw analog input values returned by **DT_BlockLoop** or **DT_ListLoop** into an array of voltage values.

Table 3: Conversion Functions (cont.)

Function Type	Function Name	Description[a]
Conversion to Raw Data Functions	DT_VoltsToDAValue	Converts a single voltage value into a raw analog output value that is used by **DT_WriteDAC.**
	DT_VoltsToDAScan	Converts an array of voltage values into a scan of raw analog output values for use with **DT_WriteDACDIO**, **DT_ScanLoop**, or **DT_FunctionGen**.
	DT_VoltsToDAChanBlock[b]	For a specified analog output channel in a block, converts an array of voltage values into a block of raw analog output values for use by **DT_BlockLoop** or **DT_ListLoop**.
	DT_VoltsToDABlock[b]	For all analog output channels in a block, converts an array of voltage values into a block of raw analog output values for use by **DT_BlockLoop** or **DT_ListLoop**.

a. For more information on **DT_ReadADC**, **DT_ReadADCDIO**, **DT_CaptureBuffer**, **DT_ScanLoop, DT_BlockLoop, DT_ListLoop, DT_WriteDAC, DT_WriteDACDIO, or DT_FunctionGen**, refer to the *DT9840 Series DSP Library User's Manual*.

b. This function is not supported in Visual Basic.

Installing the Software

The DT9840 Series Host Communication Library is installed when you install the DT9840 Series software. Refer to the *DT9840 Series Getting Started Manual* for information on installing the DT9840 Series software.

Building Host Applications Using the DT9840 Series Host Communication Library

To build a host application program using the 32-bit DT9840 Host Communication Library, reference the file dtcommlib.lib located in the following directory if you using the default installation path: Program Files\Data Translation\DT9840 Series\LIBS\dtcommlib.lib

To build a host application program using the 64-bit DT9840 Host Communication Library, reference the x64 file dtcommlib.lib located in the following directory if you using the default installation path: Program Files (x86)\Data Translation\DT9840 Series\LIBS\x64\ dtcommlib.lib

Host applications built with the 32-bit DT9840 Series Host Communication Library will run on 32-bit and 64-bit platforms. Host applications built with the 64-bit DT9840 Series Host Communication Library will run on 64-bit platforms only.

Product Support

This section provides information about obtaining support if you have trouble using the DT9840 Series Host Communication Library.

General Checklist

Should you experience problems using the DT9840 Series Host Communication Library, follow these steps:

1. Read all the appropriate sections of this manual, including any "Read This First" information.
2. Check the DT9840 Series Software CD for a README file. If present, read this file for the latest installation and usage information.
3. Check that you have installed your software properly. For information, refer to the *DT9840 Series Getting Started Manual*.
4. Check that you have installed a DT9840 Series module properly using the instructions in the *DT9840 Series Getting Started Manual*.

Note: If you are still having problems, follow the instructions provided in the next section.

Service and Support

If you have difficulty using the DT9840 Series Host Communication Library, Data Translation's Technical Support Department is available to provide technical assistance.

To request technical support, go to our web site at http://www.datatranslation.com and click on the Support link.

When requesting technical support, be prepared to provide the following information:

- Your product serial number
- The hardware/software product you need help on
- The version of the CD you are using
- Your contract number, if applicable

If you are located outside the USA, contact your local distributor; see our web site (www.datatranslation.com) for the name and telephone number of your nearest distributor.

2

Using the DT9840 Series Host Communication Library

Overview 20
Opening and Closing a Module 23
Downloading and Running a DSP Program 25
Opening and Closing a Communication Channel 26
Reading Data from a Communication Channel. 27
Writing Data to a Communication Channel. 29
Handling Channel Communication Events 31
Converting Data 32
Handling Errors 34

Overview

To communicate between the host computer and a DT9840 Series module, you need to write a host application program that calls the communication functions in the DT9840 Series Host Communication Library. In addition, you need to write a Code Composer program for the DT9840 Series module that calls the corresponding communication functions in the DT9840 Series DSP Library.

You can structure your programs in the following ways:

- **Typical Communication Programming, Host Initiated** – Write the host program to open communication channels and write the DSP program to open its end of the communication channels, as shown in Figure 2. No events are used, so you must ensure that both programs open the communication channels in the same order.

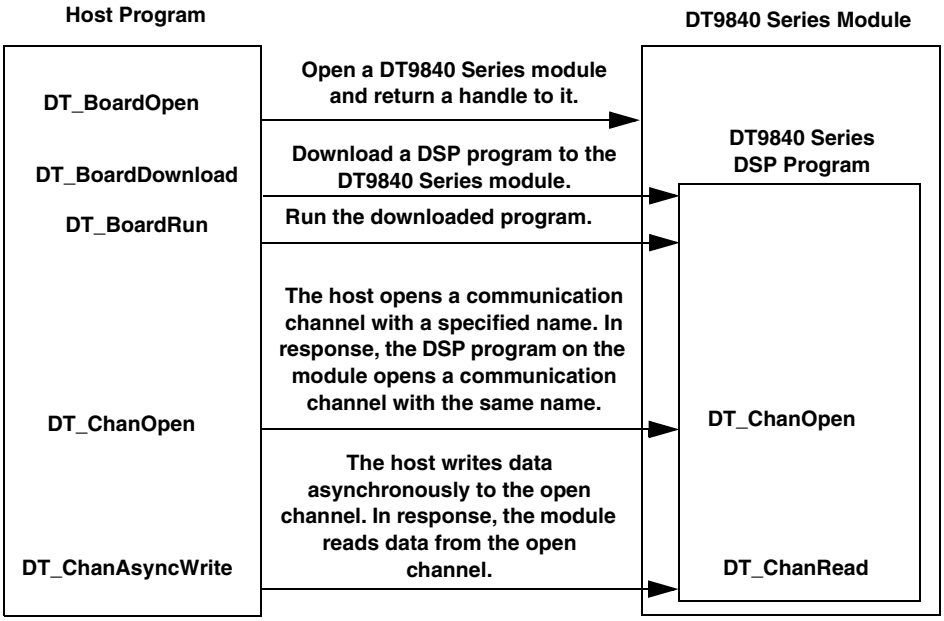

Host Program
DT9840 Series Module
DT_BoardOpen
Open a DT9840 Series module and return a handle to it.
DT_BoardDownload
Download a DSP program to the DT9840 Series module.
DT9840 Series DSP Program
DT_BoardRun
Run the downloaded program.
The host opens a communication channel with a specified name. In response, the DSP program on the module opens a communication channel with the same name.
DT_ChanOpen
DT_ChanOpen
The host writes data asynchronously to the open channel. In response, the module reads data from the open channel.
DT_ChanAsyncWrite
DT_ChanRead

Figure 2: Typical Communication Programming, Host Initiated

- **Typical Communication Programming, DSP Initiated** – Write the DSP program to open communication channels and write the host program to open its end of the communication channels, as shown in Figure 3. No events are used, so you must ensure that both programs open the communication channels in the same order.

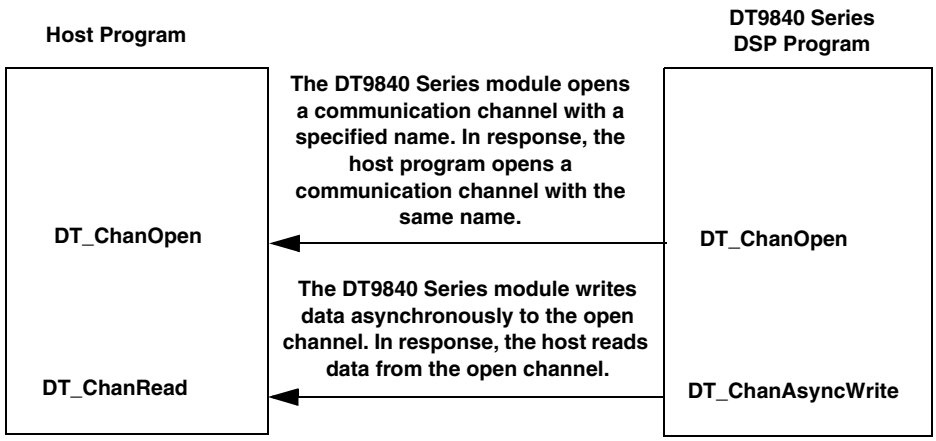

Host Program
DT9840 Series DSP Program
The DT9840 Series module opens a communication channel with a specified name. In response, the host program opens a communication channel with the same name.
DT_ChanOpen
DT_ChanOpen
The DT9840 Series module writes data asynchronously to the open channel. In response, the host reads data from the open channel.
DT_ChanRead
DT_ChanAsyncWrite

Figure 3: Typical Communication Programming, DSP Initiated

- **Event-Driven Communication Programming, Host Initiated** – Write your host program to open communication channels and write your DSP program to respond to channel open events by opening its side of communication channels, as shown in Figure 4.

  Event-driven programming provides more flexibility but is not as easy to use as typical communication programming.

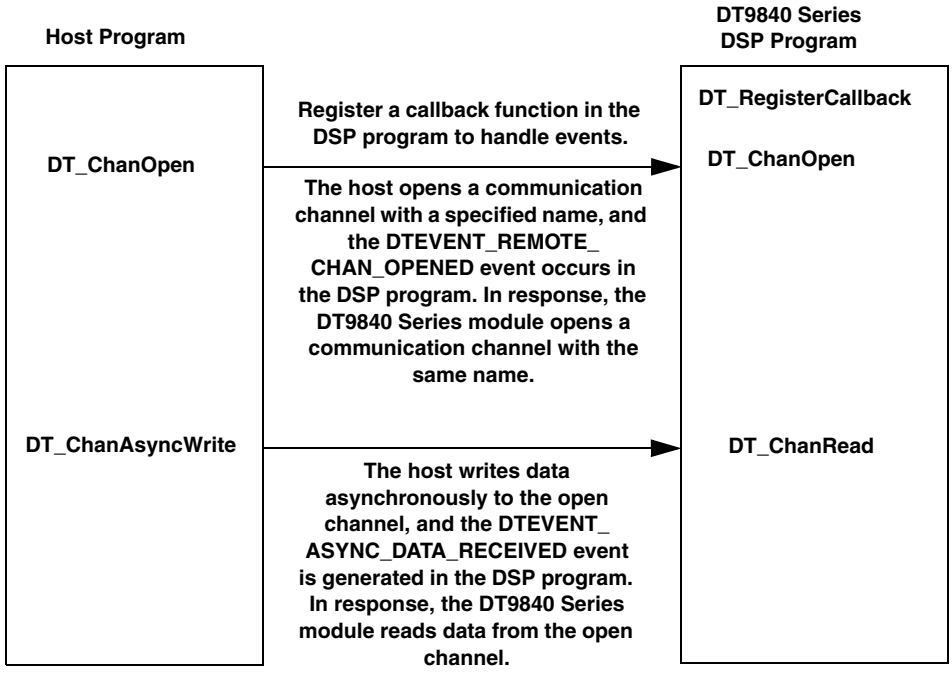

Host Program
DT9840 Series DSP Program
Register a callback function in the DSP program to handle events.
DT_RegisterCallback
DT_ChanOpen
DT_ChanOpen
The host opens a communication channel with a specified name, and the DTEVENT_REMOTE_CHAN_OPENED event occurs in the DSP program. In response, the DT9840 Series module opens a communication channel with the same name.
DT_ChanAsyncWrite
DT_ChanRead
The host writes data asynchronously to the open channel, and the DTEVENT_ASYNC_DATA_RECEIVED event is generated in the DSP program. In response, the DT9840 Series module reads data from the open channel.

Figure 4: Event-Driven Communication Programming, Host Initiated

- **Event-Driven Communication Programming, DSP Initiated** – Write your DSP program to open communication channels and write your host program to respond to channel open events by opening its side of communication channels, as shown in Figure 5.

  Event-driven programming provides more flexibility but is not as easy to use as typical communication programming.

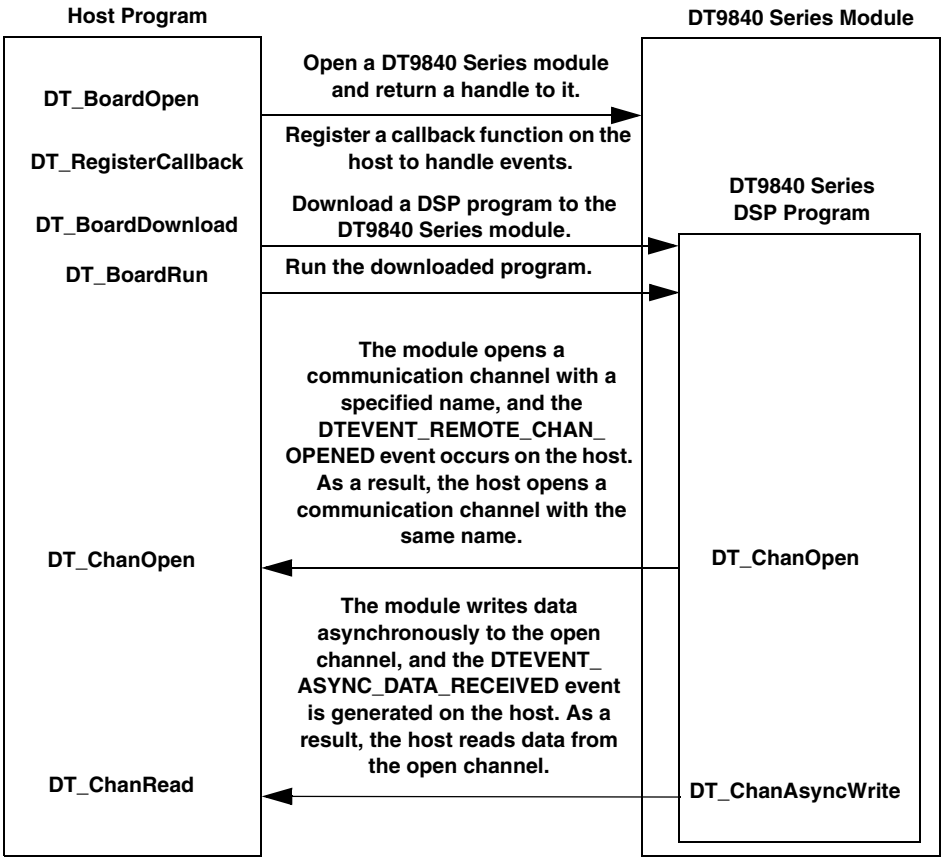

Host Program
DT9840 Series Module
DT_BoardOpen
Open a DT9840 Series module and return a handle to it.
DT_RegisterCallback
Register a callback function on the host to handle events.
DT9840 Series DSP Program
DT_BoardDownload
Download a DSP program to the DT9840 Series module.
DT_BoardRun
Run the downloaded program.
The module opens a communication channel with a specified name, and the DTEVENT_REMOTE_CHAN_OPENED event occurs on the host. As a result, the host opens a communication channel with the same name.
DT_ChanOpen
DT_ChanOpen
The module writes data asynchronously to the open channel, and the DTEVENT_ASYNC_DATA_RECEIVED event is generated on the host. As a result, the host reads data from the open channel.
DT_ChanRead
DT_ChanAsyncWrite

Figure 5: Event-Driven Communication Programming, DSP Initiated

This chapter provides conceptual information to describe the how to use the functions in the DT9840 Series Host Communication Library to write a host application program. Use this information with the reference information provided in Chapter 4 starting on page 43, for Visual C++ programmers, and in Chapter 5 starting on page 83, for Visual Basic programmers.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on using channel communication functions in your DSP program.

Opening and Closing a Module

Before you can communicate with a DT9840 Series module that is connected to the host computer, you first need to open the DT9840 Series module by calling **DT_BoardOpen** in your host application.

Note: For each module, ensure that only one host application opens and communicates with the module.

You can either specify a valid module name or specify NULL for the module name. If you specify NULL for the module name, the first DT9840 Series module that is connected to the host computer is opened. This is the simplest method of opening a module in systems where only one DT9840 Series module is installed.

If you are using multiple modules and do not know their names, you can use **DT_BoardGetName** to return the module names. You assign the module name when you configure the device driver. Refer to the *DT9840 Series Getting Started Manual* for more information on configuring the device driver.

For each DT9840 Series module that is opened, internal module buffers are created on the host that are used to store messages that are read from and written to the module. Module buffers are used in concert with channel buffers, described on page 26, to move data between the host and the module.

You can specify the size of each module input buffer (used for reading data) and output buffer (used for writing), or use the default buffer sizes (64 kBytes). A matching module buffer is created on the DT9840 Series module.

At a minimum, the size of the module input buffer must be at least as large as the largest message that the module will send to the host (using the **DT_ChanSyncWrite** or **DT_ChanAsyncWrite** function in the DSP program). The size of the module output buffer must be at least as large as the largest message that the host will send to the module (using the **DT_ChanSyncWrite** or **DT_ChanAsyncWrite** function in the host application). If the amount of memory used by your host application is not a concern, we recommend that you set the size of each module buffer to a multiple of the largest message (**DT_ChanSyncWrite** or **DT_ChanAsyncWrite**) that is sent.

Figure 6 illustrates how module buffers and channel buffers are used when the host writes data to the DT9840 Series module. Figure 7 illustrates how module buffers and channel buffers are used when the host reads data from the DT9840 Series module.

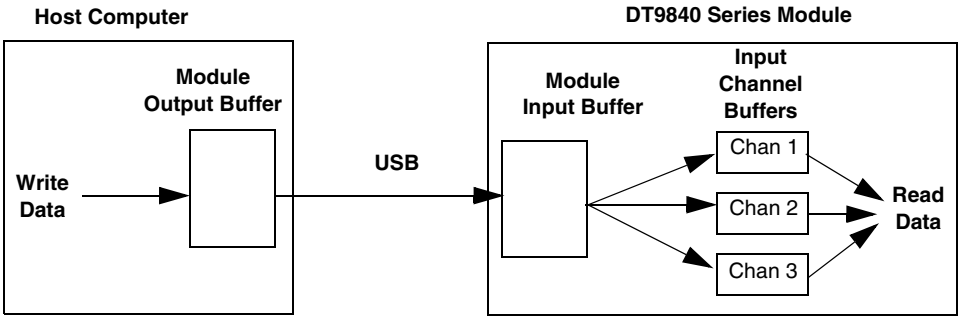


Figure 6: Module and Channel Buffers When Writing Data to a DT9840 Series Module

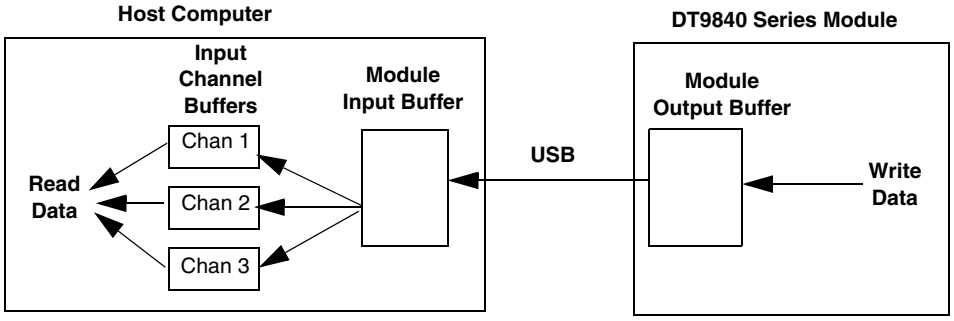


Figure 7: Module and Channel Buffers When Reading Data from a DT9840 Series Module

You can also specify the upper 16-bits of the debug flag that is passed to the DT9840 Series module using the **DT_BoardOpen** function. You can use this flag on the host to control serial debugging functions, such as printing to the serial port, on the DT9840 Series module.

DT_BoardOpen returns a handle to the DT9840 Series module that is used by all subsequent communication functions to identify the module.

When you are finished, call **DT_BoardClose** to close the module and release any resources that were allocated to it.

Downloading and Running a DSP Program

If you have not already done so, write your Code Composer program for the DT9840 Series modules using the DT9840 Series DSP Library. Code Composer creates a COFF file (Common Object File Format) with a .OUT extension that you can download to the module using the **DT_BoardDownload** function.

DT_BoardDownload resets the DSP processor on the opened DT9840 Series module and downloads the compiled DSP program over the USB bus to the DSP on the module. You can specify whether or not you want each segment that is downloaded to the DT9840 Series module to be read back from DSP memory and verified. If you choose to verify the download operation, the download rate is reduced significantly; however, verifying the download operation ensures that the program is not loaded into a non-existent memory location.

After the program is downloaded, call **DT_BoardRun** to run the DSP program on the opened DT9840 Series module.

Opening and Closing a Communication Channel

Before you can perform any channel-communication operation from the host application, such as reading from or writing to a communication channel, you first need to open the communication channel by calling **DT_ChanOpen** in your host application with a valid channel name.

DT_ChanOpen opens a unidirectional stream-based communication channel for either reading or writing, creates an input buffer if the channel was opened for reading (the size of which is user-specified), and returns a handle to the channel that is used by all subsequent communication functions to identify the channel.

Note: In general, the input buffer size should be at least as large as the largest single write of data from the DSP program on the DT9840 Series module. Refer to page 24 for more information on buffers.

You can open multiple channels between the host computer and a DT9840 Series module at any time and interleave data over the USB bus. Each channel is a unidirectional, buffered, byte stream; therefore, dealing with a communication channel from the host application program is similar to reading data from and writing data to a file.

If you are using multiple channels and do not know their names, you can use **DT_ChanGetInfo** to return the names and other information about the communication channels.

The corresponding DSP program on the module can use the same channel name to open the channel on its end and establish communication with it.

The DSP program on the DT9840 Series module must open its side of the communication channel within a specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application program to detect the timeout error and recover, if possible.

Note: When dealing with multiple channels, ensure that your host and DSP applications open the communication channels in the same order. For example, if your host application opens channels 1, 2 and 3, ensure that your DSP program opens channel 1 before it opens channel 2, and opens channels 1 and 2 before opening channel 3.

Once you have opened a communication channel, you can read data from the communication channel, as described on page 27, or write data to the communication channel, as described on page 29.

When you are finished with the communication channel, call **DT_ChanClose** to close the communication channel and release any resources that were allocated to it.

Reading Data from a Communication Channel

To read data from an opened communication channel, the DT9840 Series Host Communication Library provides the following functions:

- **DT_ChanRead** – Returns a specified number of bytes from a communication channel that was previously opened for reading. If the requested number of bytes is not yet available from the DSP program on the DT9840 Series module, this function blocks all operations until either the data is available or the function times out.

  When the host application program calls **DT_ChanRead**, the DSP program must write the specified number of bytes of data to a communication channel within the specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application to detect the timeout error and recover, if possible.

- **DT_ChanReadAvailable** – Returns the data that is available from a communication channel that was previously opened for reading, up to the size of the input channel buffer on the host. The buffer size is the maximum number of bytes that the host can read at one time. If no data is available from the DSP program on the DT9840 Series module, other operations can continue.

  When *the number of bytes that is returned* is less than the size of the channel input buffer, not all of the data from the DSP program is available; but, the data that is available is returned to the host. When *the number of bytes that is returned* is equal to the size of the channel input buffer, more than the requested data from the DSP program is available, but only the portion of data that fits in the channel input buffer is returned. The most data that can be returned at one time cannot be larger than the size of the channel input buffer.

You can return the number of bytes that are currently available in a specified channel buffer by using the **DT_ChanBytesAvailable** function.

Note that the following issues may arise when the host application program reads data from a communication channel, causing a DTEVENT_CHAN_IN_BUFFER_FULL event, described on page 209, to be generated:

- The channel input buffer is too small for the application. Increase the size of the channel input buffer by calling **DT_ChanOpen**, described on page 63, in your host application program.
- The application program was busy with another operation for too long and did not read the data from the communication channel in a timely manner. Reorganize the host application program so that reading data from the communication channel is a high priority operation.
- The host application is reading fewer bytes than the DSP program is writing to it. If the host application program is calling **DT_ChanRead** instead of **DT_ChanReadAvailable**, ensure that it is reading as many bytes as are being written by the DSP program on the module.

Note: In applications where you want to minimize the communication code in your DSP program, you can read data directly from a specified DSP address on an opened DT9840 Series module instead of reading the data from a communication channel using the **DT_BoardReadFromMemory** function.

DT_BoardReadFromMemory does not require that a DSP program is running on the module. This function operates synchronously and take precedence over all other communication operations. For example, if you use **DT_BoardReadFromMemory** to read 2 MBytes of data from the module, all other messaging between the host computer and the DSP program is suspended until all 2 MBytes has been transferred.

If you want to read a specific register on the DT9840 Series module from the host program, you can use the **DT_BoardReadRegister** function.

CAUTION:

Although you can read from any DSP address using DT_BoardReadFromMemory, it is highly recommended that you read from a known DSP address. Reading from an arbitrary DSP address can cause major system problems.

Writing Data to a Communication Channel

To write data to an opened communication channel, the DT9840 Series Host Communication Library provides the following functions:

- **DT_ChanAsyncWrite** – Asynchronously writes a specified number of bytes from an open communication channel on the host to the DT9840 Series module.

  This function writes the data and returns immediately. It does not wait for an acknowledgement from the DSP program running on the DT9840 Series module. If the communication channel is busy or backed up with data, the actual transfer to the module continues in the background; this function does not block.

  Typically, **DT_ChanAsyncWrite** is easier to use and has higher performance than **DT_ChanSyncWrite**.

- **DT_ChanSyncWrite** – Synchronously writes a specified number of bytes from an open communication channel on the host to the DT9840 Series module and waits for a response from the DSP program. This function is useful if you want to synchronize the operation of the host application with the DSP program running on the module.

  When this function is called, the data is written immediately. The function blocks all operations until the DSP application receives all the data and responds by calling **DT_ChanAcknowledge**. If the channel input buffer on the DT9840 Series module is full, the DSP program must read the specified amount of data from the channel within the specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application program to detect the timeout error and recover, if possible.

Note: If the DSP program sends a **DT_ChanSyncWrite** request to the host application program, the host application program must call **DT_ChanAcknowledge** to acknowledge that it received the request.

DT_ChanAcknowledge and **DT_ChanSyncWrite** are not supported in Visual Basic.

Note that the following issues may arise when the host application writes data to a communication channel:

- **Remote Channel Buffer Full** – If the channel input buffer on the DT9840 Series module is small or if the DSP program does not read data from the communication channel fast enough, the host can overfill the channel input buffer on the module, causing the data to back up on the communication channel.

Note: The DSP program cannot process any incoming data on any communication channel until it has read some data from the full channel input buffer.

To deal with this issue, either increase the size of the channel input buffer on module (by calling **DT_ChanOpen** in the DSP program), or increase the rate at which the DSP program reads data from the communication channel.

- **Fast Large Writes** – When the host application writes data to a communication channel, the data is first copied into the module output buffer. If the host application writes large amounts of data very quickly, the module output buffer may be filled faster than the data can be transmitted across the USB bus to the DT9840 Series module.

  To deal with this issue, either decrease the rate at which the host application program fills the module output buffer or be prepared to control how fast data is written to the channel using the **DT_ChanAsyncWrite** or **DT_ChanSyncWrite** function and the DTEVENT_OUT_BUFFER_SPACE_AVAILABLE event, described on page 213. The host application should wait until it receives a DTEVENT_OUT_BUFFER_SPACE_AVAILABLE event before attempting to write additional data to the DT9840 Series module.

Note: You can write data directly to a specified DSP address on an opened DT9840 Series module instead of writing the data to a communication channel using the **DT_BoardWriteToMemory** function.

DT_BoardWriteToMemory does not require that a DSP program is running on the module. This function operates synchronously and take precedence over all other communication operations. For example, if you use **DT_BoardWriteToMemory** to write 2 MBytes of data to the module, all other messaging between the host computer and the module is suspended until all 2 MBytes has been transferred.

If you want to write to a specific register on the DT9840 Series module from your host application, you can use the **DT_BoardWriteRegister** function.

CAUTION:

Although you can write to any DSP address using DT_BoardWriteToMemory, it is highly recommended that you write to a known DSP address. Writing to an arbitrary DSP address can cause major system problems.

To avoid cache coherency problems, the DSP and the host programs should not write to the same block of memory.

Anytime the DSP application reads data that the host wrote, the cache should be emptied first (CACHE_CLEAN). Anytime the DSP application writes data to the host, the DSP application should empty the cache (CACHE_FLUSH) after the data is written.

Handling Channel Communication Events

Various events are generated when specific actions occur in the system, such as when a channel is opened, data is received, and so on. The DT9840 Series Host Communication Library and DT9840 Series DSP Library post Windows messages that correspond to these events. Refer to Appendix C starting on page 207 for more information on events and messages.

You can write an application to trap and handle these events/messages. The host application can act as a server for the DT9840 Series module, where the host program waits for messages from the module that indicate whether the module is requesting data or needs service. The host application can then perform an appropriate action in response to each message.

The DT9840 Series Host Communication Library provides two mechanisms for monitoring asynchronous events that can be generated in your running system when using Visual C++:

- **Callback functions** – Typically, console-based programs use callback functions to monitor asynchronous events. When a specific event occurs, a registered callback function is called to notify you that the event occurred. It is the responsibility of the programmer to define this callback function so that it handles the specified events appropriately.

  Use **DT_RegisterCallback** to register the callback function. When you are finished with the callback function, use **DT_UnregisterCallback** to unregister the callback function.

Note: **DT_RegisterCallback** and **DT_UnregisterCallback** are not supported in Visual Basic.

- **Windows messaging functions** – Typically, GUI-based programs use Windows messaging functions to monitor asynchronous events. Use **DT_RegisterMsgHandler** to register the window in which all future Windows messages that are associated with specific events are received. When you are finished monitoring these events, use **DT_UnregisterMsgHandler** to unregister the message handling function.

Note: **DT_RegisterMsgHandler** and **DT_UnregisterMsgHandler** are not supported in Visual Basic.

Converting Data

The DT9840 Series Host Communication Library provides the following conversion functions for converting raw analog data to voltage values:

Note: Avoid these functions during time-critical operations, due to the extra overhead that is required.

- **DT_ADValueToVolts** – Converts a single raw analog input value that is returned by **DT_ReadADC** into a voltage value.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_ReadADC** function.

- **DT_ADScanToVolts** – Converts a scan of analog input values that is returned by **DT_ReadADCDIO**, **DT_ScanLoop**, or **DT_CaptureBuffer** into an array of voltage values.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_ReadADCDIO, DT_ScanLoop,** and **DT_CaptureBuffer** functions.

- **DT_ADBlockToVolts** – For all the analog input channels in a block, converts the raw analog input values returned by **DT_BlockLoop** or **DT_ListLoop** into an array of voltage values.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** and **DT_ListLoop** functions.

  Note: This function is not supported in Visual Basic.

- **DT_ADChanBlockToVolts** – For a specified analog input channel in a block, converts the raw analog values returned by **DT_BlockLoop** or **DT_ListLoop** into an array of voltage values.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** and **DT_ListLoop** functions.

  Note: This function is not supported in Visual Basic.

The DT9840 Series Host Communication Library provides the following conversion functions for converting voltage values to raw analog data:

Note: These functions are not recommended in time-critical operations, due to the error checking that is performed in the background.

- **DT_VoltsToDAValue** – Converts a single voltage value into a raw analog output value that is used by **DT_WriteDAC**.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_WriteDAC** function.

- **DT_VoltsToDAScan** – Converts an array of voltage values into a scan of raw analog output values for use with **DT_WriteDACDIO**, **DT_ScanLoop**, or **DT_FunctionGen**.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_WriteDACDIO**, **DT_ScanLoop**, or **DT_FunctionGen** functions.

- **DT_VoltsToDABlock** – For all analog output channels in a block, converts an array of voltage values into a block of raw analog output values for use by **DT_BlockLoop** or **DT_ListLoop**.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** or **DT_ListLoop** functions.

Note: This function is not supported in Visual Basic.

- **DT_VoltsToDAChanBlock** – For a specified analog output channel in a block, converts an array of voltage values into a block of raw analog output values for use by **DT_BlockLoop** or **DT_ListLoop**.

  Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** or **DT_ListLoop** functions.

Note: This function is not supported in Visual Basic.

Handling Errors

Each function in the DT9840 Series Host Communication Library returns a code that represents its status or error condition. Refer to Appendix A starting on page 107 for more information on each of these error codes.

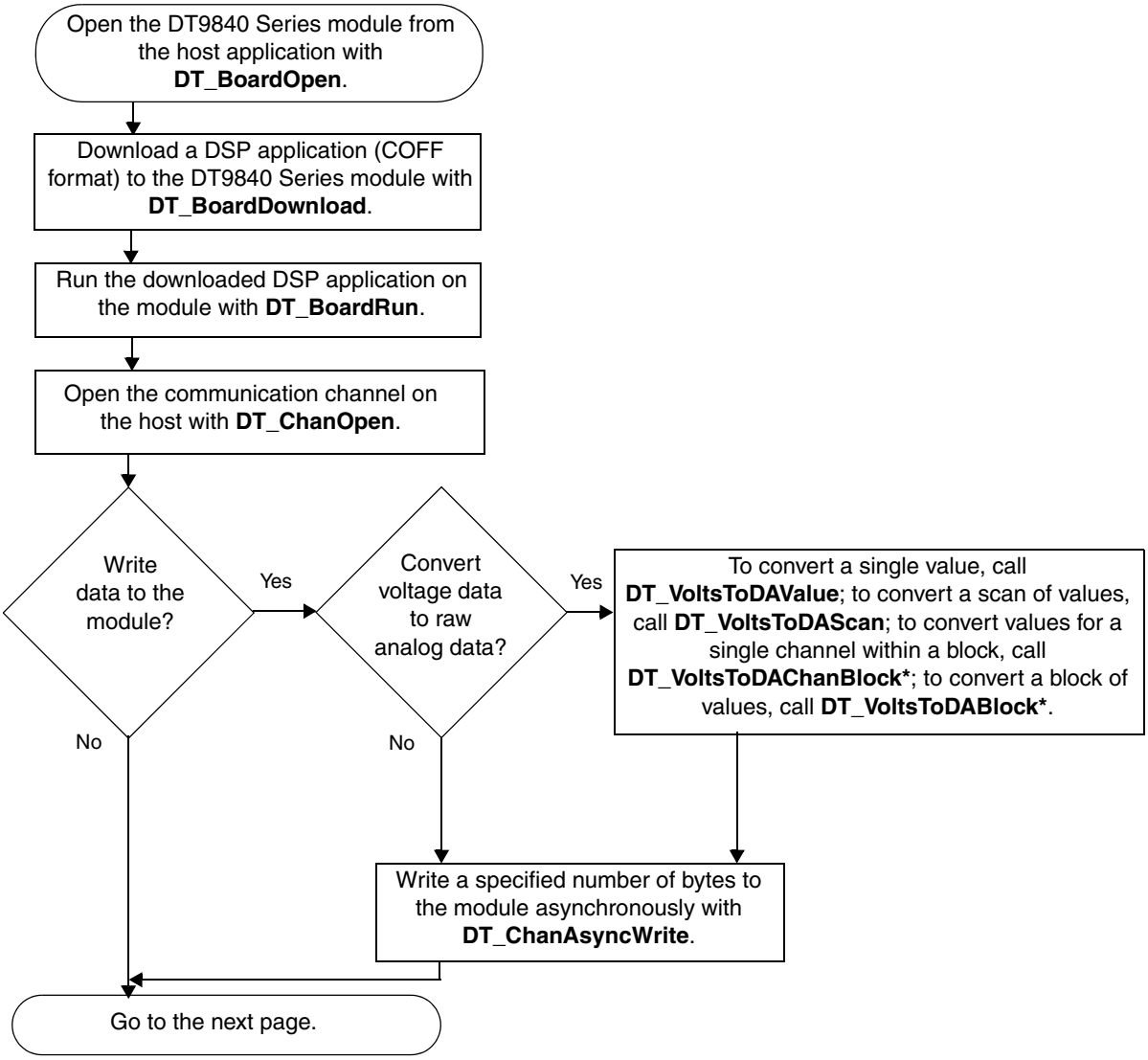
We recommend that you check the error code of each function to ensure that it was successful. If you want to return an ASCII descriptor for a specific error code, call the **DT_GetErrorString** function in your host application program.

Programming Flowcharts

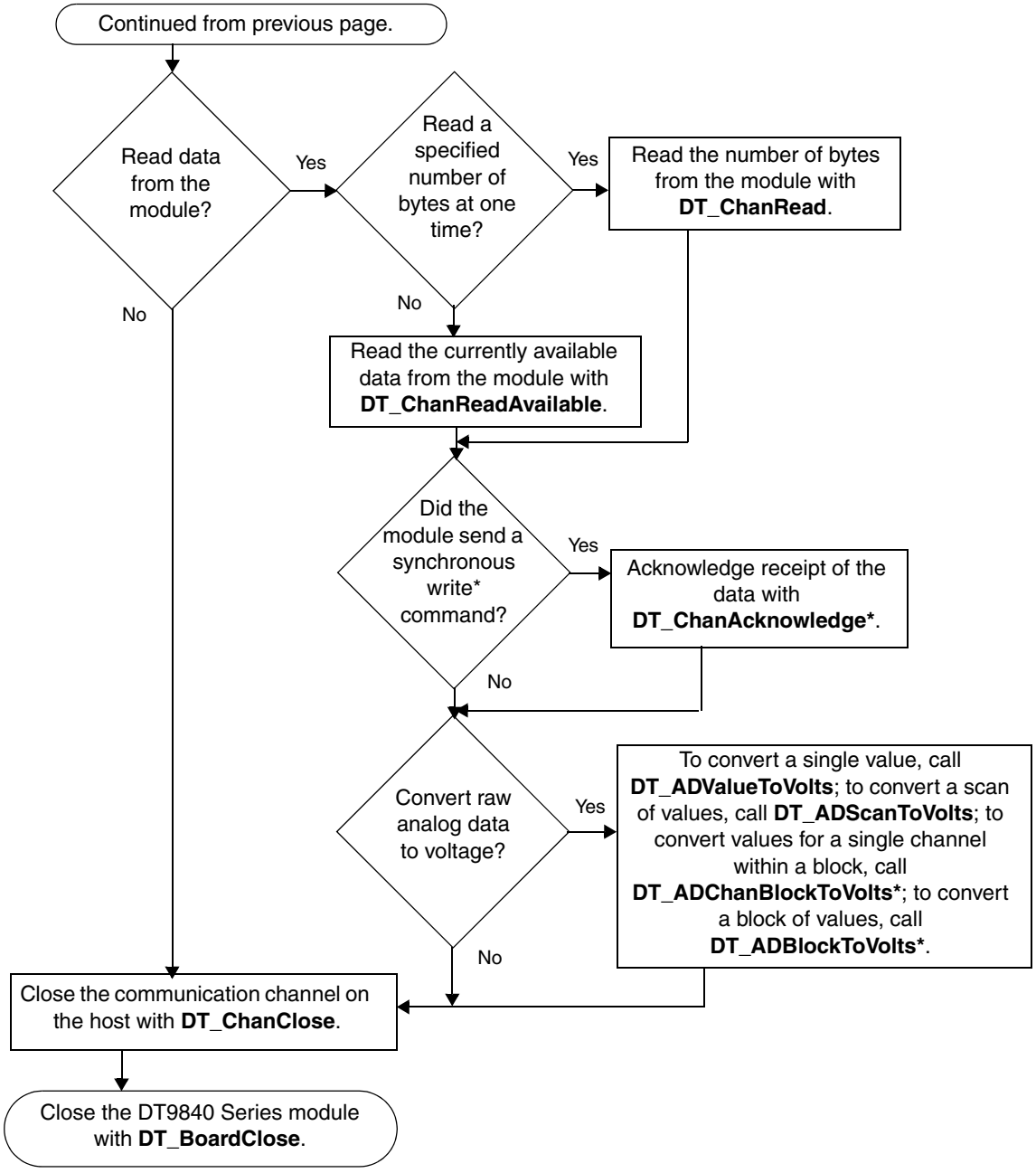
Typical Communication Operations 37
Typical Event-Driven Communication Operations. 39

The following flowcharts show the steps required to perform typical communication operations that are initiated from the host using the DT9840 Series Host Communication Library.

Typical Communication Operations


Open the DT9840 Series module from the host application with **DT_BoardOpen**.
Download a DSP application (COFF format) to the DT9840 Series module with **DT_BoardDownload**.
Run the downloaded DSP application on the module with **DT_BoardRun**.
Open the communication channel on the host with **DT_ChanOpen**.
Write data to the module?
Yes
No
Convert voltage data to raw analog data?
Yes
No
To convert a single value, call **DT_VoltsToDAValue**; to convert a scan of values, call **DT_VoltsToDAScan**; to convert values for a single channel within a block, call **DT_VoltsToDAChanBlock***; to convert a block of values, call **DT_VoltsToDABlock***.
Write a specified number of bytes to the module asynchronously with **DT_ChanAsyncWrite**.
Go to the next page.

Typical Communication Operations (cont.)

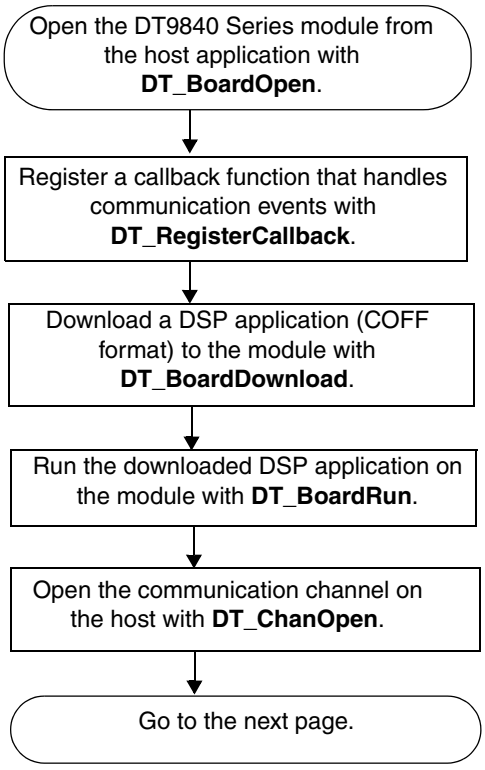

Continued from previous page.
Read data from the module?
Yes
No
Read a specified number of bytes at one time?
Yes
No
Read the number of bytes from the module with **DT_ChanRead**.
Read the currently available data from the module with **DT_ChanReadAvailable**.
Did the module send a synchronous write* command?
Yes
No
Acknowledge receipt of the data with **DT_ChanAcknowledge***.
Convert raw analog data to voltage?
Yes
No
To convert a single value, call **DT_ADValueToVolts**; to convert a scan of values, call **DT_ADScanToVolts**; to convert values for a single channel within a block, call **DT_ADChanBlockToVolts***; to convert a block of values, call **DT_ADBlockToVolts***.
Close the communication channel on the host with **DT_ChanClose**.
Close the DT9840 Series module with **DT_BoardClose**.

***DT_VoltsToDAChanBlock**, **DT_VoltsToDABlock**, **DT_ADChanBlockToVolts**, **DT_ADBlockToVolts**, and **DT_ChanAcknowledge** are not supported in Visual Basic.

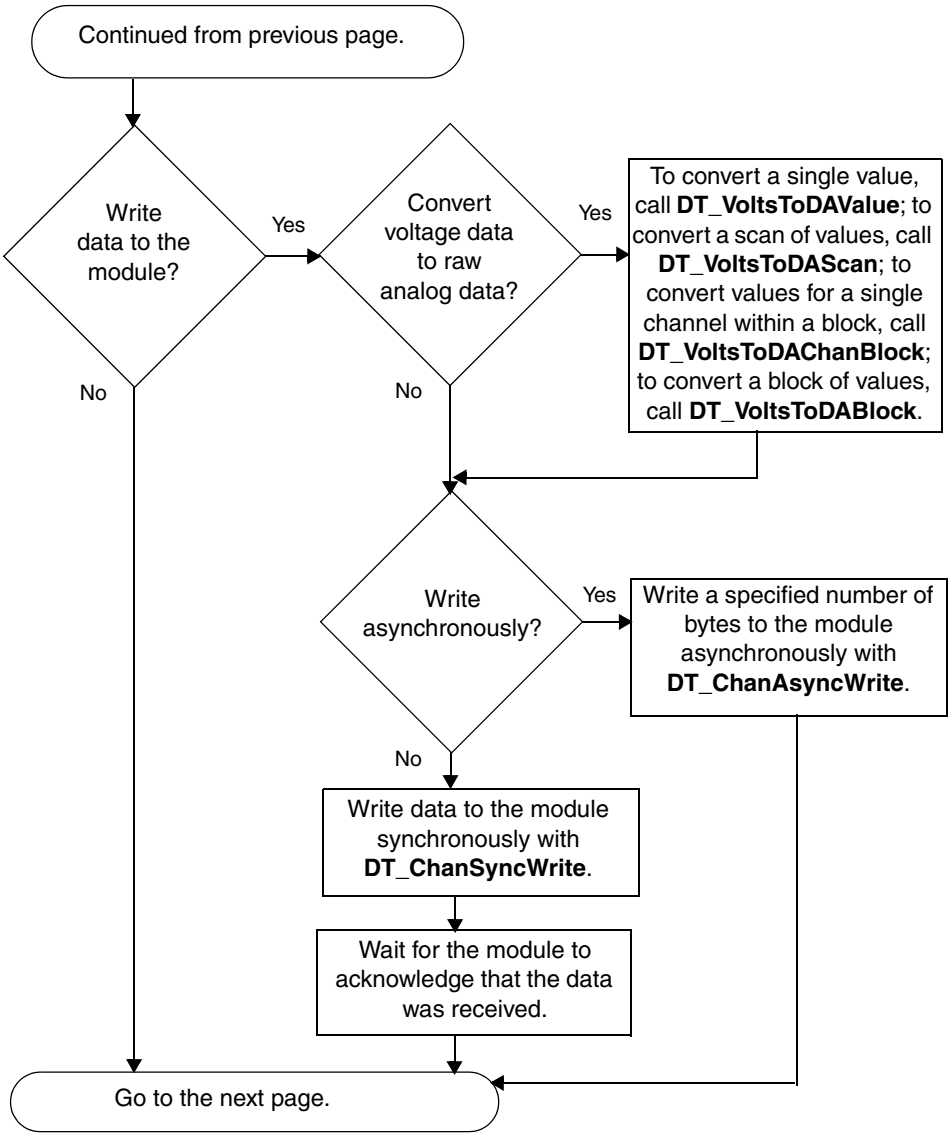
Event-driven programming is not supported in Visual Basic; therefore, do not use **DT_ChanSyncWrite** in your DSP program if you intend to write your host program using Visual Basic.

Typical Event-Driven Communication Operations

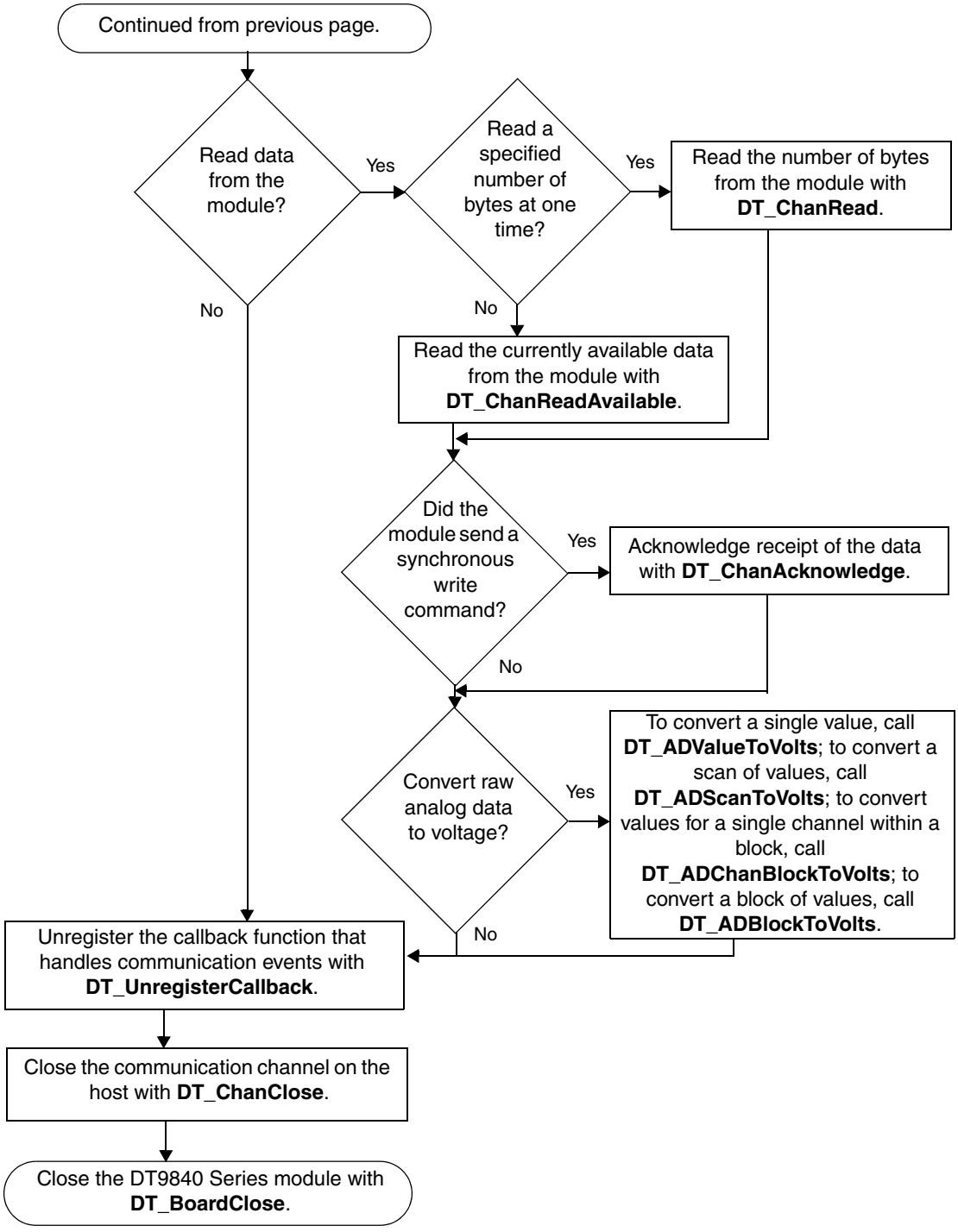
Note: Event-driven operations are not supported in Visual Basic.


Open the DT9840 Series module from the host application with **DT_BoardOpen**.
Register a callback function that handles communication events with **DT_RegisterCallback**.
Download a DSP application (COFF format) to the module with **DT_BoardDownload**.
Run the downloaded DSP application on the module with **DT_BoardRun**.
Open the communication channel on the host with **DT_ChanOpen**.
Go to the next page.

Typical Event-Driven Communication Operations (cont.)


Continued from previous page.
Write data to the module?
Yes
No
Convert voltage data to raw analog data?
Yes
No
To convert a single value, call **DT_VoltsToDAValue**; to convert a scan of values, call **DT_VoltsToDAScan**; to convert values for a single channel within a block, call **DT_VoltsToDAChanBlock**; to convert a block of values, call **DT_VoltsToDABlock**.
Write asynchronously?
Yes
No
Write a specified number of bytes to the module asynchronously with **DT_ChanAsyncWrite**.
Write data to the module synchronously with **DT_ChanSyncWrite**.
Wait for the module to acknowledge that the data was received.
Go to the next page.

Typical Event-Driven Communication Operations (cont.)


Continued from previous page.
Read data from the module?
Yes
No
Read a specified number of bytes at one time?
Yes
No
Read the number of bytes from the module with **DT_ChanRead**.
Read the currently available data from the module with **DT_ChanReadAvailable**.
Did the module send a synchronous write command?
Yes
No
Acknowledge receipt of the data with **DT_ChanAcknowledge**.
Convert raw analog data to voltage?
Yes
No
To convert a single value, call **DT_ADValueToVolts**; to convert a scan of values, call **DT_ADScanToVolts**; to convert values for a single channel within a block, call **DT_ADChanBlockToVolts**; to convert a block of values, call **DT_ADBlockToVolts**.
Unregister the callback function that handles communication events with **DT_UnregisterCallback**.
Close the communication channel on the host with **DT_ChanClose**.
Close the DT9840 Series module with **DT_BoardClose**.

4

Function Reference for Visual C++ Programmers

DT_ADBlockToVolts 45
DT_ADChanBlockToVolts 46
DT_ADScanToVolts 47
DT_ADValueToVolts 48
DT_BoardClose 49
DT_BoardDownload 50
DT_BoardGetInfo 51
DT_BoardGetName 51
DT_BoardOpen 52
DT_BoardReadFromMemory 54
DT_BoardReadRegister 55
DT_BoardRun 56
DT_BoardWriteRegister 57
DT_BoardWriteToMemory 58
DT_ChanAcknowledge 59
DT_ChanAsyncWrite 60
DT_ChanBytesAvailable 61
DT_ChanClose 61
DT_ChanGetInfo 62
DT_ChanOpen 63
DT_ChanRead 65
DT_ChanReadAvailable 67
DT_ChanSyncWrite 68
DT_GetErrorString 69
DT_RegisterCallback 70
DT_RegisterMsgHandler 73
DT_UnregisterCallback 75
DT_UnregisterMsgHandler 76
DT_VoltsToDABlock 77

DT_VoltsToDAChanBlock 78
DT_VoltsToDAScan 80
DT_VoltsToDAValue 81

This chapter provides detailed information about each of the functions in the DT9840 Series Host Communication Library for Visual C++ programmers. Functions are documented in alphabetical order.

DT_ADBlockToVolts

Syntax

```
DTSTATUS DT_ADBlockToVolts (
   INPUT_SCAN_BLOCK *pInBlock,
   ULONG NumScans,
   DOUBLE Volts[],
   ADC_RANGE VoltageRange
   );
```

Include File

DTCommonAPI.H

Description

For all the analog input channels in a block, converts the raw analog input values returned by **DT_BlockLoop** or **DT_ListLoop** into an array of voltage values.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** and **DT_ListLoop** functions.

Parameters

Name: pInBlock

Description: A pointer to an array of type INPUT_SCAN_BLOCK, described on page 155, that contains the raw analog input values to convert.

Name: NumScans

Description: An unsigned integer variable or constant that specifies the number of scans in the block to convert.

If you want to convert all of the analog input values in the block, set *NumScans* to the value of *ScansThisBlock* in the INPUT_SCAN_BLOCK structure, described on page 155.

Name: Volts[]

Description: An array of type DOUBLE in which the converted voltage values are returned.

Allocate *Volts* to hold [*NumScans*][*NUM_ADC_REGS*] values. Refer to page 117 for more information on NUM_ADC_REGS.

Name: VoltageRange

Description: A variable of type ADC_RANGE, described on page 123, that specifies the range of the A/D converter.

Notes

Avoid this function during time-critical operations, due to the overhead required.

Return Values

Appendix A starting on page 107.

Example

```
INPUT_SCAN_BLOCK *pInBlock;
DOUBLE InVolts[MAX_SCAN_SIZE] [NUM_ADC_REGS];
ULONG NumScans;
// Convert the raw AD values for all channels
// in a block into their equivalent voltage
// values in the +/- 10 V range.
// The converted values are stored in InVolts
// as an array. This example assumes that
// pInBlock points to an input block that has
// already been filled with raw AD values by
// the hardware. NumScans must be less than
// MAX_SCAN_SIZE.
NumScans = pInBlock->nScansThisBlock;
Status = DT_ADBlockToVolts(pInBlock, NumScans,
   InVolts, ADC_BIPOLAR_10_VOLTS);
```

DT_ADChanBlockToVolts

Syntax

```
DTSTATUS DT_ADChanBlockToVolts(
   INPUT_SCAN_BLOCK *pInBlock, ULONG ADChan,
   ULONG NumScans,
   DOUBLE Volts[],
   ADC_RANGE VoltageRange
   );
```

Include File

DTCommonAPI.H

Description

For a specified analog input channel in a block, converts the raw analog values returned by **DT_BlockLoop** or **DT_ListLoop** into an array of voltage values.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** and **DT_ListLoop** functions.

Parameters

Name: pInBlock

Description: A pointer to an array of type INPUT_SCAN_BLOCK, described on page 155, that contains a block of raw analog input values to convert to voltage values.

Name: ADChan

Description: An unsigned integer variable or constant that specifies on which analog input channel within the block to perform the conversion. For all DT9840 Series modules, valid values range from 0 to 7.

Name: NumScans

Description: An unsigned integer variable or constant that specifies the number of scans in the block to convert.

If you want to convert all of the values corresponding to a specified analog input channel within the block, set *NumScans* to the value of *ScansThisBlock* in the INPUT_SCAN_BLOCK structure, described on page 155.

Name: Volts[]

Description: An array of type DOUBLE in which the converted voltage values are returned.

Allocate *Volts* to hold *NumScans* values.

Name: VoltageRange

Description: A variable of type ADC_RANGE, described on page 123, that specifies the analog input voltage range.

Notes Avoid this function during time-critical operations, due to the overhead required.

Return Values See Appendix A starting on page 107.

Example

```
INPUT_SCAN_BLOCK *pInBlock;
DOUBLE InVolts[MAX_SCAN_SIZE];
ULONG NumScans;
// Convert all of the raw AD values in a block
// for a single AD channel into their
// equivalent voltage values in the +/- 10 V
// range. The converted values are stored in
// InVolts. This example assumes that pInBlock
// points to an input block that has already
// been filled with raw AD values by the
// hardware. NumScans must be less than
// MAX_SCAN_SIZE
NumScans = pInBlock->nScansThisBlock;
Status = DT_ADChanBlockToVolts(pInBlock,
   AD_CHAN_TO_CONVERT, NumScans, InVolts,
   ADC_BIPOLAR_10_VOLTS);
```

DT_ADScanToVolts

Syntax

```
DTSTATUS DT_ADScanToVolts(
   INPUT_SCAN_RCD *pInScan,
   DOUBLE Volts[],
   ADC_RANGE VoltageRange
   );
```

Include File DTCommonAPI.H

Description Converts a scan of analog input values that is returned by **DT_ReadADCDIO** or **DT_CaptureBuffer** into an array of voltage values.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_ReadADCDIO** and **DT_CaptureBuffer** functions.

Parameters

Name: pInScan

Description: A pointer to an array of type INPUT_SCAN_RCD, described on page 156, that contains the raw analog input values to convert to voltage values.

Name: Volts[]

Description: An array of type DOUBLE in which the converted voltage values are returned for each analog input channel in the scan.

Allocate the *Volts* array to hold NUM_ADC_REGS values. Refer to page 117 for more information on NUM_ADC_REGS.

Name: VoltageRange

Description: A variable of type ADC_RANGE, described on page 123, that specifies the analog input voltage range.

Notes Avoid this function during time-critical operations, due to the overhead required.

Return Values See Appendix A starting on page 107.

Example

```
INPUT_SCAN_RCD InScan;
DOUBLE InVolts[NUM_ADC_REGS];
//Convert the raw AD values for the analog
//input channels in the scan (InScan) into
//voltages in the +/- 10 V range and put
//in InVolts
Status = DT_ADScanToVolts(&InScan, InVolts,
   ADC_BIPOLAR_10_VOLTS);
```

DT_ADValueToVolts

Syntax

```
DTSTATUS DT_ADValueToVolts(
   LONG ADValue,
   DOUBLE *pVolts,
   ADC_RANGE VoltageRange
   );
```

Include File DTCommonAPI.H

Description Converts a single raw analog input value that is returned by **DT_ReadADC** into a voltage value.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_ReadADC** function.

Parameters

Name: ADValue

Description: An integer variable or constant that specifies the raw analog input value to convert into a voltage value.

Name: pVolts

Description: A pointer to a variable of type DOUBLE in which the converted voltage value is returned.

Name: VoltageRange

Description: A variable of type ADC_RANGE, described on page 123, that specifies the analog input voltage range.

Notes Avoid this function during time-critical operations, due to the overhead required.

Return Values See Appendix A starting on page 107.

Example

```
DOUBLE Volts;
LONG ADValue;
//Convert ADValue into a voltage in the
//+/- 10 V range and store in Volts
Status = DT_ADValueToVolts (ADValue, &Volts,
   ADC_BIPOLAR_10_VOLTS);
```

DT_BoardClose

Syntax

```
DTSTATUS DT_BoardClose(
   BRD_HANDLE hBoard
   );
```

Description Closes a previously opened module and releases all allocated resources that were associated with it.

Include File DTCommLib.H

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Notes None

Return Values See Appendix A starting on page 107.

Example

```
//Close the module with handle hBoard
Status = DT_BoardClose (hBoard);
```

DT_BoardDownload

Syntax

```
DTSTATUS DT_BoardDownload(
   BRD_HANDLE hBoard,
   CHAR *pFilename,
   BOOL bVerifyFlag
   );
```

Include File

DTCommLib.H

Description

Downloads a compiled DSP program over the USB bus to the DSP processor of the DT9840 Series module. This function supports the standard output format (COFF file - Common Object File Format - with a .OUT extension) of Code Composer Studio for compiled programs.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: pFilename

Description: A pointer to a variable of type CHAR that specifies the name of the COFF file (Common Object File Format - with a .OUT extension) to download to the DT9840 Series module.

Name: bVerifyFlag

Description: A Boolean variable that specifies whether or not to verify the download operation.

Set *bVerifyFlag* to TRUE if you want each segment that is downloaded to the module to be read back from DSP memory and verified. Setting this parameter to TRUE slows down the download significantly, but ensures that the DSP program is not loaded into a non-existent memory location.

Set *bVerifyFlag* to FALSE if you do not want to verify the download operation.

Notes

This function resets the DSP processor before downloading the program. After the program is downloaded, call **DT_BoardRun**, described on page 56, to run the program.

Return Values

See Appendix A starting on page 107.

Example

```
BRD_HANDLE hBoard;
//Download the file LEDflash.out to the module
//and do not verify the download operation
Status = DT_BoardDownload (hBoard,
   "LEDflash.out", FALSE);
```

DT_BoardGetInfo

Syntax

```
DTSTATUS DT_BoardGetInfo(
   BRD_HANDLE hBoard,
   BRD_INFO *pBoardInfo
   );
```

Include File DTCommLib.H

Description Returns information about the connected DT9840 Series module, including its memory configuration.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: pBoardInfo

Description: A pointer to variable of type BRD_INFO, described on page 143, in which the type, name, state, memory configuration, and FPGA version of the specified DT9840 Series module is returned.

Notes Ensure that TCF file for your module has the appropriate settings. Refer to the *DT9840 Series DSP Library User's Manual* for more information about TCF files.

Return Values See Appendix A starting on page 107.

Example

```
// Return info about the DT9840 Series module
BRD_INFO pBoardInfo;
Status = DT_BoardGetInfo (hBoard,
   &pBoardInfo);
```

DT_BoardGetName

Syntax

```
DTSTATUS DT_BoardGetName(
   WORD BoardIndex,
   BOARD_NAME BoardName
   );
```

Include File DTCommLib.H

Description Returns the name of a connected DT9840 Series module.

Parameters

Name: BoardIndex

Description: A variable of type WORD that specifies the index of the DT9840 Series module whose name you want to return. The first DT9840 Series module installed in the system has a *BoardIndex* of 0.

Name: BoardName

Description: A variable of type BOARD_NAME, described on page 116, in which the ASCII name of the DT9840 Series module is returned.

Notes

This function is rarely needed. It is used only in very generalized cases, where for example, a host-based application requires the names of all connected DT9840 Series modules at runtime.

More typical programs deal with a single DT9840 Series module with a well known name. In these cases, the host application can open the module directly by name with **DT_BoardOpen**, described on page 52.

Return Values

See Appendix A starting on page 107.

Example

```
// Print the names of all DT9840 Series
// modules installed in the system.
BOARD_NAME BoardName;
int BoardIndex;
BoardIndex = 0;
While (DT_BoardGetName (BoardIndex, BoardName)
   == DT_SUCCESS)
{
  Printf (Board #%d name = %s\n, BoardIndex,
   BoardName);
  BoardIndex++;
}
```

DT_BoardOpen

Syntax

```
DTSTATUS DT_BoardOpen(
   CHAR *BoardName,
   BRD_HANDLE *phBoard,
   BOARD_OPTIONS *pBoardOptions
   );
```

Include File

DTCommLib.H

Description

Opens a DT9840 Series module by name, sets the size the module input and output buffers, and returns a handle to the module. This handle is then used by all subsequent communication functions to identify the module.

Parameters

Name: BoardName

Description: A pointer to a variable of type CHAR that specifies the name of the DT9840 Series module. You assign this name when you configure the device driver.

If you are using a single DT9840 Series module, you can specify NULL for *BoardName* to open the module. If you are using multiple DT9840 Series modules, you must specify the actual name of the module.

Refer to the *DT9840 Series Getting Started Manual* for more information on assigning a module name.

Name: phBoard

Description: A pointer to a variable of type BRD_HANDLE, described on page 116, in which the handle to the open DT9840 Series module is returned. If the module cannot be opened, NULL is returned.

Name: pBoardOptions

Description: A pointer to a variable of type BOARD_OPTIONS, described on page 141, that specifies the size of the module input and output buffers.

Specify NULL for *pBoardOptions* to set the module input and output buffers to their default sizes (64 kBytes). This is the normal operating mode.

Notes

Only one host application should be communicating with a module.

If you are using multiple modules and do not know their names, you can use **DT_BoardGetName**, described on page 51, to return the module names.

When you are finished with the module, call **DT_BoardClose**, described on page 49, to close the module and release any resources that were allocated to it.

Return Values

See Appendix A starting on page 107.

Example

```
CHAR BdName;
BRD_HANDLE hBoard;
BOARD_OPTIONS BrdOptions;
BrdOptions.SizeThisStructure =
   sizeof(BOARD_OPTIONS);
//Specify a module input buffer size of
//64 kBytes
BrdOptions.ReadSizeInKb = 64;
//Specify a module output buffer size of
//200 kBytes
BrdOptions.WriteBufSizeInKb = 200;
```

Example (cont.)

```
//Open module BdName, set the internal buffer
//sizes, and return the handle in hBoard
Status = DT_BoardOpen (BdName, &hBoard,
   &BrdOptions);
```

DT_BoardReadFromMemory

Syntax

```
DTSTATUS DT_BoardReadFromMemory(
   BRD_HANDLE hBoard,
   ULONG SrcDSPAddress,
   ULONG NumBytes,
   VOID *pDestPCBuffer
   );
```

Include File DTCommLib.H

Description Returns data from a DSP address on the specified DT9840 Series module into a variable on the host computer (with no assistance from the DSP).

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: SrcDSPAddress

Description: An unsigned integer variable or constant that specifies the DSP address on the DT9840 Series module that you want to read.

Name: NumBytes

Description: An unsigned integer variable or constant that specifies the number of bytes to read from the DSP on the DT9840 Series module.

Name: pDestPCBuffer

Description: A pointer to a variable of type VOID on the host in which the data from the DSP on the DT9840 Series module is returned.

Notes

Although you can read from any DSP address using this function, it is highly recommended that you read from a known DSP address. Reading from an arbitrary DSP address can cause major system problems.

This function operates synchronously and take precedence over all other communication operations. For example, if you use this function to read 2 MBytes of data from the DT9840 Series module, all other messaging between the host computer and the module is suspended until all 2 MBytes has been transferred.

Unlike most other functions in this library, **DT_BoardReadFromMemory** does not require that a DSP program is running on the module.

To read specific registers on the DT9840 Series module, use **DT_BoardReadRegister**, described on page 55.

Return Values

See Appendix A starting on page 107.

Example

```
BRD_HANDLE hBoard;
ULONG DSPAdd;
ULONG HostValue;
//Return 50 bytes from DSPAdd on the module to
//HostValue on the host computer
Status = DT_BoardReadFromMemory (hBoard,
   DSPAdd, 50, &HostValue);
```

DT_BoardReadRegister

Syntax

```
DTSTATUS DT_BoardReadRegister(
   BRD_HANDLE hBoard,
   ULONG Register,
   ULONG *pValue
   );
```

Include File

DTCommLib.H

Description

Reads the value of a specific register on a DT9840 Series module.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: Register

Description: An unsigned integer variable or constant that specifies the register on the DT9840 Series module to read.

Name: pValue

Description: A pointer to an unsigned integer variable in which the value of the specified register on the DT9840 Series module is returned.

Notes The DTCommonAPI.H include file defines constants that you can use for *Register*. Refer to page 116 for more information on these constants.

Return Values See Appendix A starting on page 107.

Example

```
BRD_HANDLE BrdHandle;
ULONG *pValue;
//Read the value of the register for analog
//input channel 0
Status = DT_BoardReadRegister (BrdHandle,
   AD0_IN_REG, &pValue);
```

DT_BoardRun

Syntax

```
DTSTATUS DT_BoardRun(
   BRD_HANDLE hBoard,
   CHAR *pArgStr
   );
```

Include File DTCommLib.H

Description Runs a DSP program that was downloaded previously using **DT_BoardDownload**, described on page 50.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: pArgStr

Description: Support for this argument is currently not implemented; specify NULL for *pArgStr*.

A pointer to a variable of type CHAR that contains new arguments for the program. This string is parsed and passed to the main procedure of the DSP program. Specify NULL for *pArgStr* if you have no new arguments for the DSP program.

Notes When a DSP program is downloaded (using **DT_BoardDownload**, described on page 50), the program is loaded into DSP memory but is not started. **DT_BoardRun** starts program execution.

If a DSP program is already running when this function is called, **DT_BoardRun** resets the DSP, passes new arguments to the program, and re-runs the program.

Return Values See Appendix A starting on page 107.

Example

```
//On module hBoard, run the previously
//downloaded program, changing none of the
//program arguments.
Status = DT_BoardRun (hBoard, NULL);
```

DT_BoardWriteRegister

Syntax

```
DTSTATUS DT_BoardWriteRegister(
   BRD_HANDLE hBoard,
   ULONG Register,
   ULONG Value
   );
```

Include File DTCommLib.H

Description Writes a value to a specified register on a DT9840 Series module.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: Register

Description: An unsigned integer variable or constant that specifies the register on the DT9840 Series module to write to.

Name: pValue

Description: An unsigned integer variable that specifies the value to write to the specified register on the DT9840 Series module.

Notes The DTCommonAPI.H include file defines constants that you can use for *Register*. Refer to page 116 for more information on these constants.

Return Values See Appendix A starting on page 107.

Example

```
//Write a value of 0 to the AD0 input register
BRD_HANDLE BrdHandle;
Status = DT_BoardWriteRegister (BrdHandle,
   AD0_IN_REG, 0);
```

DT_BoardWriteToMemory

Syntax

```
DTSTATUS DT_BoardWriteToMemory(
   BRD_HANDLE hBoard,
   VOID *pSrcPCBuffer,
   ULONG NumBytes,
   ULONG DestDSPAddress
   );
```

Include File

DTCommLib.H

Description

Writes data from a variable on the host computer to an address in DSP memory on a DT9840 Series module (with no assistance from the DSP).

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: pSrcPCBuffer

Description: A pointer to a variable of type VOID that contains the data from the host to write to the DSP on a DT9840 Series module.

Name: NumBytes

Description: An unsigned integer variable or constant that specifies the number of bytes to write to the DSP on a DT9840 Series module.

Name: DestDSPAddress

Description: An unsigned integer variable or constant that specifies the address of DSP memory on the DT9840 Series module to write to.

Notes

Although you can write to any DSP address using this function, it is highly recommended that you write to a known DSP address. Writing to an arbitrary DSP address can cause major system problems.

This function operates synchronously and take precedence over all other communication operations. For example, if you use this function to write 2 MBytes of data from the host computer to the DSP of a DT9840 Series module, all other messaging between the host computer and the module is suspended until all 2 MBytes has been transferred.

Unlike most other functions in this library, **DT_BoardWriteToMemory** does not require that a DSP program is running on the DT9840 Series module.

To write to specific registers on the DT9840 Series module, use **DT_BoardWriteRegister**, described on page 57.

Return Values See Appendix A starting on page 107.

Example

```
BRD_HANDLE hBoard;
ULONG DSPAdd;
VOID HostVars;
//Write all the values in HostVars on the host
//computer to DSPAdd on the module
Status = DT_BoardWriteToMemory (hBoard,
   &HostVars, sizeof (HostVars), DSPAdd);
```

DT_ChanAcknowledge

Syntax

```
DTSTATUS DT_ChanAcknowledge(
   CHAN_HANDLE hChan
   );
```

Include File DTCommonAPI.H

Description Acknowledges that the host application received a synchronous write command from a DT9840 Series module.

Parameters

Name: hChan

Description: A variable of type CHAN_HANDLE, described on page 116, that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 63.

Notes When the DT9840 Series module writes data synchronously to the host application, the module writes a specified number of bytes to an opened communication channel and waits for a response from the host application. When all the data has been received by the host, the host application should respond by calling this function.

This function is useful if you want to synchronize the operation of the host application and the DSP program on the module.

Return Values See Appendix A starting on page 107.

Example

```
//Acknowledge a synchronous write command from
//the module on the open communication channel
//hChan
Status = DT_ChanAcknowledge(hChan);
```

DT_ChanAsyncWrite

Syntax

```
DTSTATUS DT_ChanAsyncWrite(
   CHAN_HANDLE hChan,
   VOID *pBuffer,
   ULONG NumBytes
   );
```

Include File

DTCommonAPI.H

Description

Asynchronously writes a specified number of bytes from an open communication channel on the host to a DT9840 Series module.

Parameters

Name: hChan

Description: A variable of type CHAN_HANDLE, described on page 116, that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 63.

Name: pBuffer

Description: A pointer to a variable of type VOID that contains the data to write to the DT9840 Series module.

Name: NumBytes

Description: An unsigned integer variable or constant that specifies the number of bytes to write to the DT9840 Series module.

Notes

This function writes the data and returns immediately. It does not wait for an acknowledgement from the DT9840 Series module. If the communication channel is busy or backed up with data, the actual transfer to the DT9840 Series module continues in the background; this function does not block.

If you want to write data to the DT9840 Series module asynchronously and wait for an acknowledgement from the module that the data was received, use **DT_ChanSyncWrite**, described on page 68, instead.

Return Values

See Appendix A starting on page 107.

Example

```
#define BUFF_SIZE (8*1024)
CHAN_HANDLE hChan;
CHAR ChBuff[BUFF_SIZE+1];
//Write 100 bytes from ChBuff on the host to
//the module
Status = DT_ChanAsyncWrite(hChan, &ChBuff,
   100);
```

DT_ChanBytesAvailable

Syntax

```
DTSTATUS DT_ChanBytesAvailable(
   CHAN_HANDLE hChan,
   ULONG *pBytesAvailable
   );
```

Include File DTCommonAPI.H

Description Returns the number of bytes that are currently available in the specified channel buffer.

Parameters

Name: hChan

Description: A variable of type CHAN_HANDLE, described on page 116, that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 63.

Name: pBytesAvailable

Description: A pointer to an unsigned integer variable in which the number of bytes from the specified channel buffer is returned.

Return Values See Appendix A starting on page 107.

Example

```
CHAN_HANDLE hChan;
ULONG *NumBytesAvailable;

//Read the number of bytes available and put
//it in NumBytesAvailable
Status = DT_ChanBytesAvailable (hChan,
   &NumBytesAvailable);
```

DT_ChanClose

Syntax

```
DTSTATUS DT_ChanClose(
   CHAN_HANDLE hChan
   );
```

Include File DTCommonAPI.H

Description Closes a previously opened communication channel on the host and releases all allocated resources that were associated with it.

Parameters

Name: hChan

Description: A variable of type CHAN_HANDLE, described on page 116, that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 63.

Notes None

Return Values See Appendix A starting on page 107.

Example

```
//Close channel hChan on the host
Status = DT_ChanClose (hChan);
```

DT_ChanGetInfo

Syntax

```
DTSTATUS DT_ChanGetInfo(
   BRD_HANDLE hBoard,
   WORD ChanIndex,
   CHAN_NAME ChanName,
   CHAN_TYPE *pChanType,
   ULONG *pHostBufSize,
   ULONG *pDSPBufSize
   );
```

Include File DTCommonAPI.H

Description Returns information about the channel that is associated with the specified *ChanIndex*.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: ChanIndex

Description: A variable of type WORD that specifies the index of the communication channel. The first open communication channel has a *ChanIndex* of 0.

Name: ChanName

Description: A variable of type CHAN_NAME, described on page 116, in which the ASCII name of the active communication channel is returned.

Name: pChanType

Description: A pointer to a variable of CHAN_TYPE, described on page 124, in which the type of operation for the communication channel is returned.

Name: pHostBufSize

Description: A pointer to an integer variable in which the size of the channel input buffer on the host computer is returned.

Name: pDSPBufSize

Description: A pointer to an integer variable in which the size of the channel input buffer on the DT9840 Series module is returned.

Notes

This function is rarely needed. It is used only in very generalized cases, where for example, an application requires the names of all active channels at runtime.

More typical programs deal with specific channels with well known names. In these cases, the application can open the channel directly by name using **DT_ChanOpen**, described on page 63.

Return Values

See Appendix A starting on page 107.

Example

```
//Get the information for all
//communication channels on the
//host and print it
BRD_HANDLE hBoard;
CHAN_NAME ChanName;
CHAN_TYPE ChanType;
ULONG HostBufSize;
ULONG DSPBufSize;
WORD Index;
CHAR *ChanTypeStr;
Index = 0;
While (DT_ChanGetInfo (hBoard, Index,
   ChanName, &ChanType, &HostBufSize,
   &DSPBufSize) == DT_SUCCESS)
{
  if (ChanType == READ_CHAN)
    ChanTypeStr = "READ";
  else
    ChanTypeStr = "WRITE";
    Printf ("Channel #%d name= %s Type=%s
   DSPBufSize=%d\n", Index, ChanName,
   ChanTypeStr, DSPBufSize);
    Index++;
}
```

DT_ChanOpen

Syntax

```
DTSTATUS DT_ChanOpen(
   BRD_HANDLE hBoard,
   CHAR *pChanName,
   CHAN_TYPE ChanType,
   ULONG BufSize,
   CHAN_HANDLE *phChan,
   ULONG TimeoutInMS
   );
```

Include File

DTCommonAPI.H

Description Opens a unidirectional stream-based communication channel by name, creates an input buffer for the channel if it was opened for reading, and returns a handle to that channel. This handle is then used by all subsequent channel communication functions to identify the channel.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: pChanName

Description: A pointer to a NULL-terminated CHAR array that specifies the name of the communication channel to open. The maximum length of this array is determined by CHAN_NAME_MAX_LEN, described on page 116.

Name: ChanType

Description: A variable of type CHAN_TYPE, described on page 124, that specifies the type of operation to perform on the communication channel.

Name: BufSize

Description: An unsigned integer variable or constant that specifies the size, in bytes, of the input buffer for a communication channel that was opened for reading.

If *ChanType* = READ_CHAN, *BufSize* determines how much data the communication channel on the host computer can receive from the DT9840 Series module at one time. In general, the size of the channel input buffer should be at least as large as the largest single write of data from the module.

Note that a channel input buffer that is larger than needed has no negative performance impact on I/O operations.

If *ChanType* = WRITE_CHAN, *BufSize* is ignored on the host, but is passed to the DT9840 Series module. Therefore, you could use this value to indicate to the DSP program how large to create the matching channel input buffer on the module, if you wish.

Name: phChan

Description: A pointer to a variable of type CHAN_HANDLE, described on page 116, in which the handle to the opened communication channel is returned. If this function fails, *phChan* points to a NULL value.

Name: TimeoutInMS

Description: An unsigned integer variable or constant that specifies the timeout value, in milliseconds.

If *TimeoutInMS* = 0, the timeout period is disabled.

Notes Once the communication channel is open, you can use the other channel functions to send data to, receive data from, and manage the communication channel.

A host application can open up to 32 arbitrarily named channels and write data to or read data from them. The corresponding program on the DT9840 Series module can then use the channel names to open matching channels and establish communication with them.

The DT9840 Series module must open its side of the channel within the specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application to detect the timeout error and recover, if possible.

When dealing with multiple channels, ensure that your host and DSP applications open the communication channels in the same order. For example, if your host application opens channels 1, 2 and 3, ensure that your DSP program opens channel 1 before it opens channel 2, and opens channels 1 and 2 before opening channel 3.

When you are finished with it, ensure that you close the communication channel using **DT_ChanClose**, described on page 61, to release any resources that were associated with it.

Return Values See Appendix A starting on page 107.

Example

```
BRD_HANDLE bHandle;
CHAN_HANDLE hChan;
//Open "chan1" on the host for reading with a
//input buffer size of 8992 bytes and return
//the handle to the channel in hChan. If the
//DSP program does not open its side of the
//channel in 10000 ms, time out.
Status = DT_ChanOpen(bHandle, (CHAR*)"chan1",
   READ_CHAN, 8992, &hChan, 10000);
```

DT_ChanRead

Syntax

```
DTSTATUS DT_ChanRead(
   CHAN_HANDLE hChan,
   VOID *pBuffer,
   ULONG NumBytes,
   ULONG TimeoutInMS
   );
```

Include File DTCommonAPI.H

Description Returns a specified number of bytes from a previously opened communication channel.

Parameters

Name: hChan

Description: A variable of type CHAN_HANDLE, described on page 116, that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 63.

Name: pBuffer

Description: A pointer to a variable of type VOID in which the data from the DT9840 Series module is returned.

Name: NumBytes

Description: An unsigned integer variable or constant that specifies the number of bytes to read from the DT9840 Series module.

Name: TimeoutInMS

Description: An unsigned integer variable or constant that specifies the timeout value, in milliseconds.

If *TimeoutInMS* = 0, the timeout period is disabled.

Notes If the requested number of bytes is not yet available from the DT9840 Series module, this function blocks all operations until either the data is available or the function times out. If you prefer a non-blocking operation, use **DT_ChanReadAvailable**, described on page 67.

When the host application calls **DT_ChanRead**, the DSP application on the module must write *NumBytes* bytes of data to the communication channel within the specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application to detect the timeout error and recover, if possible.

Return Values See Appendix A starting on page 107.

Example

```
CHAN_HANDLE hChan;
VOID *HostBuff;
//Read 100 bytes from the module and put it in
//HostBuff. If the module does not write the
//specified number of bytes in 1000 ms, a
//timeout occurs.
Status = DT_ChanRead (hChan, &HostBuff, 100,
   1000);
```

DT_ChanReadAvailable

Syntax

```
DTSTATUS DT_ChanReadAvailable(
   CHAN_HANDLE hChan,
   VOID *pBuffer,
   ULONG BufSize
   ULONG *pBytesRead
   );
```

Include File DTCommonAPI.H

Description Returns the data that is available from the DT9840 Series module, up to the size of the channel input buffer *(BufSize*).

Parameters

Name: hChan

Description: A variable of type CHAN_HANDLE, described on page 116, that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 63.

Name: pBuffer

Description: A pointer to a variable of type VOID in which the data from the DT9840 Series module is returned.

Name: BufSize

Description: An unsigned integer variable or constant that specifies the size of the *pBuffer* variable on the host computer, in bytes. This value is the maximum number of bytes that can be read from the DT9840 Series module and stored in *pBuffer* at one time.

Name: pBytesRead

Description: A pointer to an unsigned integer variable in which the number of bytes that were read is returned; this value can never be greater than *BufSize*.

When *the number that is returned* is less than *BufSize*, not all of the data from the DT9840 Series module is available; but, the data that is available is returned to the host application.

When *the number that is returned* is equal to *BufSize*, more than the requested data from the DT9840 Series module is available, but only the portion of data that fits in *pBuffer* is returned.

Notes The most data that can be returned at one time cannot be larger than *BufSize*.

When this function is called, the available data, up to the size of *pBuffer*, is returned. If no data is available, other operations can continue.

If you want to block other operations from continuing while you are waiting for all the data to become available, use the **DT_ChanRead** function, described on page 65, instead.

Return Values See Appendix A starting on page 107.

Example

```
CHAN_HANDLE hChan;
VOID *HostBuff;
ULONG *BytesRead;
//Read the number of bytes available from the
/ module up to (100 bytes), and put it in
//HostBuff. Return the actual number of bytes
//read in BytesRead
Status = DT_ChanReadAvailable (hChan,
   &HostBuff, 100, &BytesRead);
```

DT_ChanSyncWrite

Syntax

```
DTSTATUS DT_ChanSyncWrite(
   CHAN_HANDLE hChan,
   VOID *pBuffer,
   ULONG NumBytes,
   ULONG TimeoutInMS
   );
```

Include File DTCommonAPI.H

Description Synchronously writes a specified number of bytes from an open communication channel on the host computer to the DT9840 Series module and waits for a response from the module.

Parameters

Name: hChan

Description: A variable of type CHAN_HANDLE, described on page 116, that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 63.

Name: pBuffer

Description: A pointer to a variable of type VOID that contains the data to write to the DT9840 Series module.

Name: NumBytes

Description: An unsigned integer variable or constant that specifies the number of bytes to write to the DT9840 Series module.

Name: TimeoutInMS

Description: An unsigned integer variable or constant that specifies the timeout value, in milliseconds.

If *TimeoutInMS* = 0, the timeout period is disabled.

Typically, **DT_ChanSyncWrite** executes immediately with no delay. The *TimeoutInMS parameter* is useful in the unlikely event that the channel input buffer on the DT9840 Series module becomes full and the function blocks operations while waiting for room to be available in the channel input buffer.

Notes This function blocks all operations until the DT9840 Series module receives all the data and responds by calling **DT_ChanAcknowledge**, described on page 59.

When this function is called, the data is written immediately.

If the channel input buffer on the DT9840 Series module is full, the module must read the specified amount of data from the channel within the specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application to detect the timeout error and recover, if possible.

This function is useful if you want to synchronize the operation of the host application with the DSP application on the module. Typically, **DT_ChanAsyncWrite**, described on page 59, is easier to use and has higher performance than **DT_ChanSyncWrite**.

Return Values See Appendix A starting on page 107.

Example

```
CHAN_HANDLE hChan;
CHAR ChBuff[8993];
//Write 100 bytes from ChBuff to the module
//and wait for a response from the module.
//Timeout if a response is not received in
//10000 ms
Status = DT_ChanSyncWrite(hChan, &ChBuff, 100,
   10000);
```

DT_GetErrorString

Syntax

```
CHAR* DT_GetErrorString (
   DTSTATUS Status,
   CHAR* pErrString
   );
```

Include File DTCommonAPI.H

Description Returns an ASCII string that corresponds to a specific error code.

Parameters

Name: Status

Description: An unsigned integer variable or constant that specifies the error code for which to return a string descriptor.

Name: pErrString

Description: A pointer to a variable of type CHAR in which a copy of the string that is returned by this function is stored. If NULL, the returned ASCII string is not copied.

Notes All of the functions in the DT9840 Series DSP Library return error codes that you can use with this function.

Return Values A pointer to an variable of type CHAR in which the ASCII string is returned. See Appendix A starting on page 107 for more information on the error codes that can be returned or detected by the DT9840 Series Host Communication Library.

Example

```
//Utility for reporting DTSTATUS errors
DTSTATUS CheckErr (DTSTATUS Status)
{
  CHAR *pErrStr;
  if (Status)
   {
     pErrStr = DT_GetErrorString (Status,
      NULL);
     LOG_printf(&trace, pErrStr);
     SYS_exit(-1);
   }
  else
     return (Status);
   }
}
```

DT_RegisterCallback

Syntax

```
DTSTATUS DT_RegisterCallback(
   BRD_HANDLE hBoard,
   CALLBACKPROC pCallback,
   ULONG EventTraps,
   VOID *pContext
   );
```

Include File DTCommonAPI.H

Description Registers a callback function that is called whenever an event occurs on the DT9840 Series module.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

If you specify a valid value for *hBoard*, the registered event handler is called for events that occur on the associated DT9840 Series module. If you specify NULL for *hBoard*, all events regardless of the module are sent to the registered event handler.

Name: pCallback

Description: A pointer to a callback function of type CALLBACKPROC, described on page 162, that is called whenever an event occurs on the DT9840 Series module.

Name: EventTraps

Description: An unsigned integer variable that specifies the type of event about which to be notified. The following constants are defined:

- DTEVENT_REMOTE_CHAN_OPENED, described on page 215.
- DTEVENT_REMOTE_CHAN_CLOSED, described on page 214.
- DTEVENT_ASYNC_DATA_RECEIVED, described on page 208.
- DTEVENT_SYNC_DATA_RECEIVED, described on page 215.
- DTEVENT_CHAN_IN_BUFFER_FULL, described on page 209.
- DTEVENT_OUT_BUFFER_SPACE_AVAILABLE, described on page 213.
- DTEVENT_ERROR, described on page 210.
- DTEVENT_AD_OVERRUN, described on page 208; this event occurs in the DSP program on the DT9840 Series module only.
- DTEVENT_DA_UNDERRUN, described on page 210; this event occurs in the DSP program on the DT9840 Series module only.
- DTEVENT_MODULE_UNPLUGGED, described on page 212; this event can occur in the DSP or host program.

Description (cont.):

- DTEVENT_MODULE_PLUGGED, described on page 211; this event can occur in a DSP program only.
- DTEVENT_ALL_COMM_EVENTS. Specify this constant to monitor all of the host communication events.

To register for multiple events, OR the appropriate constants together.

Name: pContext

Description: A pointer to a variable of type VOID that is passed to a user-defined callback function when **DT_RegisterCallback** is called.

A typical use for *pContext* is to pass a "this" pointer in C++ so that the callback function has access to the object that registered the callback.

Notes

Various events are generated when specific actions occur in the system (such as when a module is opened, a message is received, and so on). The DT9840 Series Host Communication Library and DT9840 Series DSP Library post messages that correspond to these events. You can write your host application program to respond to these events.

If you call **DT_RegisterCallback** and specify the events to trap, a user-specified callback function is called whenever one of these events occurs. Refer to Appendix C starting on page 207 for more information on events and messages.

When you are finished with the callback function, use **DT_UnregisterCallback**, described on page 75, to unregister the callback function.

As an alternative to callback functions, you can use a Windows message handler to get information related to the module. Use **DT_RegisterMsgHandler**, described on page 73, to register a Windows message handler.

Return Values

See Appendix A starting on page 107.

Example

```
//Register a callback to handle channel open
//requests from the specified module (hBoard)
DT_RegisterCallback(hBoard,
   RemoteChanOpenProc,
   DTEVENT_REMOTE_CHAN_OPENED, NULL);
//RemoteChanOpenProc is invoked when the
//module opens a channel. The EventCode is
//DTEVENT_REMOTE_CHAN_OPENED; pContext is
//context information passed in
//DT_RegisterCallback; Param1 is not used;
//Param2 is a pointer to the CHAN_OPEN_INFO
//structure; the return value is ignored
```

Example (cont.)

```
BOOL RemoteChanOpenProc(IN_PARAM ULONG
   EventCode, IN_PARAM VOID *pContext,
   IN_PARAM LONG Param1, IN_PARAM LONG Param2)
{
  CHAN_OPEN_INFO *pChanInfo = (CHAN_OPEN_INFO
   *)Param2;
  switch (pChanInfo-> RemoteChanType)
  {
  case WRITE_CHAN:
    {
      //open a channel to read data
      DT_ChanOpen(NULL, pChanInfo->ChanName,
       READ_CHAN, pChanInfo->RemoteBufSize,
       &g_hChanRead, TIMEOUT_OPEN)
    }
    break;
  case READ_CHAN:
   {
   //open a channel to write data
   DT_ChanOpen(NULL, pChanInfo->ChanName,
     WRITE_CHAN, pChanInfo->RemoteBufSize,
     &g_hChanWrite, TIMEOUT_OPEN)
   }
   break;
  }
//when all the channels have been opened, this
//callback can be unregistered
}
```

DT_RegisterMsgHandler

Syntax

```
DTSTATUS DT_RegisterMsgHandler(
   BRD_HANDLE hBoard,
   HWND hWindow,
   ULONG EventTraps
   );
```

Include File DTCommLib.H

Description Registers the window in which future Windows messages that are associated with a DT9840 Series module are displayed.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Description (cont.): If you specify a valid value for *hBoard*, the registered event handler is called for events that occur on the associated DT9840 Series module. If you specify NULL for *hBoard*, all events regardless of the module are sent to the registered event handler.

Name: hWindow

Description: A variable of type HWND that specifies the handle to the window in which all future Windows messages that are associated with the specified DT9840 Series module are displayed.

Name: EventTraps

Description: An unsigned integer variable that specifies the type of event about which to be notified. The following constants are defined:

- DTEVENT_REMOTE_CHAN_OPENED, described on page 215.
- DTEVENT_REMOTE_CHAN_CLOSED, described on page 214.
- DTEVENT_ASYNC_DATA_RECEIVED, described on page 208.
- DTEVENT_SYNC_DATA_RECEIVED, described on page 215.
- DTEVENT_CHAN_IN_BUFFER_FULL, described on page 209.
- DTEVENT_OUT_BUFFER_SPACE_AVAILABLE, described on page 213.
- DTEVENT_ERROR, described on page 210.
- DTEVENT_AD_OVERRUN, described on page 208; this event occurs in the DSP program on the DT9840 Series module only.
- DTEVENT_DA_UNDERRUN, described on page 210; this event occurs in the DSP program on the DT9840 Series module only.
- DTEVENT_MODULE_UNPLUGGED, described on page 212; this event can occur in either the host or DSP program.
- DTEVENT_MODULE_PLUGGED, described on page 211; this event can occur in a DSP program only.
- DTEVENT_ALL_COMM_EVENTS. Specify this constant to monitor all of the host communication events.

To register for multiple events, OR the appropriate constants together.

Notes

Various events are generated when specific actions occur in the system (such as when a module is opened, a message is received, and so on). The DT9840 Series Host Communication Library and DT9840 Series DSP Library post messages that correspond to these events. You can write your host application program to respond to these events.

If you call **DT_RegisterMsgHandler** with a valid window handle and specify the events to trap, an appropriate Windows message is sent to the registered window when the event occurs. Refer to Appendix C starting on page 207for detailed information on each message that can be returned.

When you are finished with the Windows message handler, use **DT_UnregisterMsgHandler**, described on page 76, to unregister the window.

As an alternative to Windows messages, you can use a callback function to get information related to the module. Use **DT_RegisterCallback**, described on page 70, to register a callback function.

Return Values

See Appendix A starting on page 107.

Example

```
BRD_HANDLE hBoard;
HWND hWinHandle;
//Register a window handle in which a message
//will be posted when the DSP has opened its
//side of the communication channel in
//response to a request from the host
Status = DT_RegisterMsgHandler(hBoard,
   hWinHandle, DTEVENT_REMOTE_CHAN_OPENED);
```

DT_UnregisterCallback

Syntax

```
DTSTATUS DT_UnregisterCallback(
   BRD_HANDLE hBoard,
   CALLBACKPROC pCallback
   );
```

Description

Unregisters a callback function that was previously registered using the **DT_RegisterCallback** function, described on page 70.

Include File

DTCommonAPI.H

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: pCallback

Description: A pointer to a user-defined callback function of type CALLBACKPROC, described on page 162, that you want to unregister.

Notes None

Return Values See Appendix A starting on page 107.

Example

```
//Unregister the callback function
//RemoteChanOpenProc
DTStatus = DT_UnregisterCallback(hBoard,
   RemoteChanOpenProc);
```

DT_UnregisterMsgHandler

Syntax

```
DTSTATUS DT_UnregisterMsgHandler(
   BRD_HANDLE hBoard,
   HWND hWindow
   );
```

Include File DTCommLib.H

Description Unregisters the message handler window that was previously registered using the **DT_RegisterMsgHandler** function, described on page 73.

Parameters

Name: hBoard

Description: A variable of type BRD_HANDLE, described on page 116, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 52.

Name: hWindow

Description: A variable of type HWND that specifies the message window that you want to unregister.

Notes None

Return Values See Appendix A starting on page 107.

Example

```
HWND hWinHandle;
//Unregister the message handler for the
//specified window (hWinHandle) and module
//(hBoard)
Status = DT_UnregisterMsgHandler(hBoard,
   hWinHandle);
```

DT_VoltsToDABlock

Syntax

```
DTSTATUS DT_VoltsToDABlock(
   DOUBLE Volts[],
   ULONG NumScans,
   OUTPUT_SCAN_BLOCK *pOutBlock,
   DAC_RANGE VoltageRange
   );
```

Include File

DTCommonAPI.H

Description

For all analog output channels in a block, converts an array of voltage values into a block of raw analog output values for use by **DT_BlockLoop** or **DT_ListLoop**.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** or **DT_ListLoop** functions.

Parameters

Name: Volts[]

Description: An array of type DOUBLE that contains the voltage values to convert into raw analog output values.

Allocate *Volts* to hold [*NumScans*][*NUM_DAC_CHANS]* values. For the DT9841, DT9841E, DT9841-VIB, and DT9842/2, the value of NUM_DAC_CHANS is 2. For the DT9842/8, the value of NUM_DAC_CHANS is 8.

Name: NumScans

Description: An unsigned integer variable or constant that represents the number of scans for each block.

If you want to convert all of the analog output values in the block, set *NumScans* to the value of *ScansThisBlock* in the OUTPUT_SCAN_BLOCK structure, described on page 158.

Name: pOutBlock

Description: A pointer an array of type OUTPUT_SCAN_BLOCK, described on page 158, in which the raw analog output values are returned.

Name: VoltageRange

Description: A variable of type DAC_RANGE, described on page 131, that specifies the analog output voltage range.

Notes

Avoid this function during time-critical operations, due to the overhead required.

Return Values

See Appendix A starting on page 107.

Example

```
INPUT_SCAN_BLOCK *pOutBlock;
DOUBLE OutVolts[MAX_SCAN_SIZE]
   [NUM_DAC_CHANS];
ULONG NumScans;
```

Example (cont.)

```
// Converts an array of output voltages in the
// +/- 10 V range to equivalent raw DA values
// and stores them in an Output Block as the
// values for both DA channels. This example
// assumes that pOutBlock points to an output
// block that needs to be filled with raw DA
// values before being sent to the hardware.
// NumScans must be less than MAX_SCAN_SIZE
NumScans = pOutBlock->nScansThisBlock;
// Generate a ramp from 0 to 10 volts on Chan 0
// Generate a ramp from 10 to 0 volts on Chan 1
for (i=0; i< NumScans; i++)
{
  OutVolts[i][0] = (10.0 * i) / NumScans;
  OutVolts[i][1] = 10.0 - (10.0 * i) /
   NumScans;
}
Status = DT_VoltsToDABlock(OutVolts, NumScans,
   pOutBlock, DAC_BIPOLAR_10_VOLT);
```

DT_VoltsToDAChanBlock

Syntax

```
DTSTATUS DT_VoltsToDAChanBlock( DOUBLE
   Volts[],
   ULONG DAChan,
   ULONG NumScans,
   OUTPUT_SCAN_BLOCK *pOutBlock,
   DAC_RANGE VoltageRange
   );
```

Include File DTCommonAPI.H

Description For a specified analog output channel in a block, converts an array of voltage values into a block of raw analog output values for use by **DT_BlockLoop** or **DT_ListLoop**.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_BlockLoop** or **DT_ListLoop** functions.

Parameters

Name: Volts[]

Description: An array of type DOUBLE that contains the voltage values to convert into raw analog output values.

Allocate *Volts* to hold *NUM_DAC_CHANS* values. For the DT9841, DT9841E, DT9841-VIB, and DT9842/2, the value of NUM_DAC_CHANS is 2. For the DT9842/8, the value of NUM_DAC_CHANS is 8.

Name: DAChan

Description: An unsigned integer variable or constant that specifies on which analog output channel within the block to perform the conversion. For the DT9841, DT9841E, DT9841-VIB, and DT9842/2, valid values are 0 and 1. For the DT9842/8, valid values are 0 to 7.

Name: NumScans

Description: An unsigned integer variable or constant that specifies the number of scans in each block to convert.

If you want to convert all of the values in the block corresponding to a specified analog output channel, set *NumScans* to the value of *ScansThisBlock* in the OUTPUT_SCAN_BLOCK structure, described on page 158.

Name: pOutBlock

Description: A pointer to an array of type OUTPUT_SCAN_BLOCK, described on page 158, in which the raw analog output values are returned.

Name: VoltageRange

Description: A variable of type DAC_RANGE, described on page 131, that specifies the analog output voltage range.

Notes Avoid this function during time-critical operations, due to the overhead required.

Return Values See Appendix A starting on page 107.

Example

```
OUTPUT_SCAN_BLOCK *pOutBlock;
DOUBLE OutVolts[MAX_SCAN_SIZE];
ULONG NumScans;
// Converts an array of output voltages in the
// +/-10 V range to equivalent raw DA values
// and stores them in an Output Block as the
// values for one DA channel. This example
// assumes that pOutBlock points to an output
// block that needs to be filled with raw DA
// values before being sent to the hardware.
// NumScans must be less than MAX_SCAN_SIZE
NumScans = pOutBlock->nScansThisBlock;
// Generate a ramp from 0 to 10 volts
for (i=0; i< NumScans; i++)
OutVolts[i] = (10.0 * i) / NumScans;
// Convert data to raw DA values in pOutBlock
Status = DT_VoltsToDAChanBlock(OutVolts,
   DA_CHAN_TO_CONVERT, NumScans, pOutBlock,
   DAC_BIPOLAR_10_VOLT);
```

DT_VoltsToDAScan

Syntax

```
DTSTATUS DT_VoltsToDAScan(
   DOUBLE Volts[],
   OUTPUT_SCAN_RCD *pOutScan,
   DAC_RANGE VoltageRange
   );
```

Include File

DTCommonAPI.H

Description

Converts an array of voltage values into a scan of raw analog output values for use with **DT_WriteDACDIO**, **DT_ScanLoop**, or **DT_FunctionGen**.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_WriteDACDIO**, **DT_ScanLoop**, or **DT_FunctionGen** functions.

Parameters

Name: Volts[]

Description: An array of type DOUBLE that contains the voltage values to convert into raw analog output values.

Allocate *Volts* to hold NUM_DAC_CHANS voltage values; for the DT9841, DT9841E, DT9841-VIB, and DT9842/2, the value of NUM_DAC_CHANS is 2. For the DT9842/8, the value of NUM_DAC_CHANS is 8.

Name: pOutScan

Description: A pointer to an array of type OUTPUT_SCAN_RCD, described on page 160, in which the raw analog output values are returned.

Name: VoltageRange

Description: A variable of type DAC_RANGE, described on page 131, that specifies the analog output voltage range.

Notes

Avoid this function during time-critical operations, due to the overhead required.

Return Values

See Appendix A starting on page 107.

Example

```
OUTPUT_SCAN_RCD OutScan;
DOUBLE OutVolts[NUM_DAC_CHANS];
//Convert the voltage values, in the +/- 10 V
//range, for the scan (OutVolts) into raw
//values and put in OutScan.
Status = DT_VoltsToDAScan(OutVolts, &OutScan,
   DAC_BIPOLAR_10_VOLT);
```

DT_VoltsToDAValue

Syntax

```
DTSTATUS DT_VoltsToDAValue(
   DOUBLE Volts,
   LONG *pDAValue,
   DAC_RANGE VoltageRange
   );
```

Include File

DTCommonAPI.H

Description

Converts a single voltage value into a raw analog output value that is used by **DT_WriteDAC**.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_WriteDAC** function.

Parameters

Name: Volts

Description: A constant or variable of type DOUBLE that contains the voltage value to convert into a raw analog output value.

Name: pDAValue

Description: A pointer to an integer variable in which the converted raw analog output value is returned.

Name: VoltageRange

Description: A variable of type DAC_RANGE, described on page 131, that specifies the analog output voltage range.

Notes

Avoid this function during time-critical operations, due to the overhead required.

Return Values

See Appendix A starting on page 107.

Example

```
DOUBLE Volts;
LONG Value;
//Convert the voltage in Volts, which is in
//the +/- 10 V range, into a raw value and
//store in Value
Status = DT_VoltsToDAValue (Volts, &Value,
   DAC_BIPOLAR_10_VOLT);
```

5

Function Reference for Visual Basic Programmers

DT_ADScanToVolts 84
DT_ADValueToVolts 85
DT_BoardClose 85
DT_BoardDownload 86
DT_BoardGetInfo 87
DT_BoardGetName 88
DT_BoardOpen 89
DT_BoardReadFromMemory 90
DT_BoardReadRegister 91
DT_BoardRun 92
DT_BoardWriteRegister 93
DT_BoardWriteToMemory 94
DT_ChanAsyncWrite 95
DT_ChanBytesAvailable 96
DT_ChanClose 97
DT_ChanGetInfo 98
DT_ChanOpen 99
DT_ChanRead 101
DT_ChanReadAvailable 103
DT_GetErrorString 104
DT_VoltsToDAScan 105
DT_VoltsToDAValue 106

This chapter provides detailed information about each of the functions in the DT9840 Series Host Communication Library for Visual Basic programmers. The functions are documented in alphabetical order.

DT_ADScanToVolts

Syntax

```
Declare Function DT_ADScanToVolts Lib
   "DTCommLib.dll" (
   ByRef InScan As INPUT_SCAN_RCD,
   ByRef Volts As Double,
   ByVal VoltageRange As ADC_RANGE
   ) As DTSTATUS
```

Public Interface

DTCommLib.BAS

Description

Converts a scan of analog input values that is returned by **DT_ReadADCDIO** or **DT_CaptureBuffer** into an array of voltage values.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_ReadADCDIO** and **DT_CaptureBuffer** functions.

Parameters

Name: InScan

Description: A variable of type INPUT_SCAN_RCD, described on page 204, that contains the raw analog input values to convert to voltage values.

Name: Volts

Description: A variable of type DOUBLE in which the converted voltage values are returned for each analog input channel in the scan.

Allocate the *Volts* array to hold NUM_ADC_REGS values. Refer to page 164 for more information on NUM_ADC_REGS.

Name: VoltageRange

Description: A variable of type ADC_RANGE, described on page 170, that specifies the analog input voltage range.

Notes

Avoid this function during time-critical operations, due to the overhead required.

Return Values

See Appendix A starting on page 107.

Example

```
'Convert the raw AD values for all the analog
'input channels in the scan (InScan) into
'voltages in the +/- 10 V range and put in
'InVoltsArray
Dim Status As DTSTATUS
Dim InScan As INPUT_SCAN_RCD
Dim InVoltsArray(NUM_ADC_REGS) As Double

Status = DT_ADScanToVolts(InScan,
   InVoltsArray(0), BIP10_VOLTS_SPAN)
```

DT_ADValueToVolts

Syntax

```
Declare Function DT_ADValueToVolts Lib
   "DTCommLib.dll" (
   ByVal ADValue As Long,
   ByRef Volts As Double,
   ByVal VoltageRange As ADC_RANGE
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Converts a single raw analog input value that is returned by **DT_ReadADC** into a voltage value.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_ReadADC** function.

Parameters

Name: ADValue

Description: A variable of type LONG that specifies the raw analog input value to convert into a voltage value.

Name: Volts

Description: A variable of type DOUBLE in which the converted voltage value is returned.

Name: VoltageRange

Description: A variable of type ADC_RANGE, described on page 170, that specifies the analog input voltage range.

Notes Avoid this function during time-critical operations, due to the overhead required.

Return Values See Appendix A starting on page 107.

Example

```
'Convert Value into a voltage in the +/- 10 V
'range and store in Volts
Dim Status As DTSTATUS
Dim Value As Long
Dim Volts As Double

Status = DT_ADValueToVolts(Value, Volts,
   BIP10_VOLTS_SPAN)
```

DT_BoardClose

Syntax

```
Declare Function DT_BoardClose
   Lib "DTCommLib.dll" (
   ByVal hBoard As Long
   ) As DTSTATUS
```

Description Closes a previously opened module and releases all allocated resources that were associated with it.

Public Interface DTCommLib.BAS

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Notes None

Return Values See Appendix A starting on page 107.

Example

```
'Close the module with handle BoardHandle
Dim Status As DTSTATUS
Dim BoardHandle As Long

Status = DT_BoardClose(BoardHandle)
```

DT_BoardDownload

Syntax

```
Declare Function DT_BoardDownload Lib
   "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByVal FileName As String,
   ByVal bVerify As Boolean
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Downloads a compiled DSP program over the USB bus to the DSP processor on the DT9840 Series module. This function supports the standard output format (COFF file) of Code Composer Studio for compiled programs.

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: Filename

Description: A fixed length STRING variable that specifies the name of the DSP (COFF) file to download to the DT9840 Series module.

Name: bVerify

Description: A variable of type BOOLEAN that specifies whether or not to verify the download operation.

Description (cont.): Set *bVerify* to TRUE if you want each segment that is downloaded to the DT9840 Series module to be read back from DSP memory and verified. Setting this parameter to TRUE slows down the download significantly, but ensures that the DSP program is not loaded into a non-existent memory location.

Set *bVerify* to FALSE if you do not want to verify the download operation.

Notes This function resets the DSP processor before downloading the program. After the program is downloaded, call **DT_BoardRun**, described on page 92, to run the program.

Return Values See Appendix A starting on page 107.

Example

```
'Download the file "LEDflash.out" to the
'module and do not verify the download
'operation
Dim Status As DTSTATUS
Dim BoardHandle As Long

Status = DT_BoardDownload( BoardHandle,
   "LEDflash.out", False)
```

DT_BoardGetInfo

Syntax

```
Declare Function DT_BoardGetInfo Lib
   "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByRef BoardStatus As BRD_INFO
   ) As DTSTATUS
```

Include File DTCommLib.H

Description Returns information about the connected DT9840 Series module, including its memory configuration.

Parameters

Name: hBoard

Description: A variable of type LONG, that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: BoardStatus

Description: A variable of type BRD_INFO, described on page 190, in which the type, name, state, memory configuration, and FPGA version of the specified DT9840 Series module is returned.

Notes Ensure that TCF file for your module has the appropriate settings. Refer to the *DT9840 Series DSP Library User's Manual* for more information about TCF files.

Return Values See Appendix A starting on page 107.

Example

```
' Return the type of DT9840 Series module
' installed in the system.
Dim Status As DTSTATUS
Dim BoardHandle As BRD_HANDLE
Dim BoardInfo As BRD_INFO

Status = DT_BoardGetName (BoardHandle,
   BoardInfo)
```

DT_BoardGetName

Syntax

```
Declare Function DT_BoardGetName Lib
   "DTCommLib.dll" (
   ByVal BoardIndex As Long,
   ByVal BoardName As String
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Returns the name of a connected DT9840 Series module.

Parameters

Name: BoardIndex

Description: A variable of type LONG the specifies the index of the DT9840 Series module whose name you want to return. The first DT9840 Series module installed in the system has a *BoardIndex* of 0.

Name: BoardName

Description: A fixed length STRING variable in which the ASCII name of the DT9840 Series module is returned.

Notes This function is rarely needed. It is used only in very generalized cases, where for example, a host-based application requires the names of all connected DT9840 Series modules at runtime.

More typical programs deal with a single module with a well known name. In these cases, the host application can open the module directly by name with **DT_BoardOpen**, described on page 89.

Return Values See Appendix A starting on page 107.

Example

```
' Get the name of the first DT9840 Series
' module installed in the system, and store it
' in the string BoardName
Dim Status As DTSTATUS
Dim BoardIndex As Long
Dim BoardName As String * 256
BoardIndex = 0

Status = DT_BoardGetName (BoardIndex,
   BoardName)
```

DT_BoardOpen

Syntax

```
Declare Function DT_BoardOpen
   Lib "DTCommLib.dll" (
   ByVal BoardName As String,
   ByRef hBoard As Long,
   ByRef BoardOptions As Any
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Opens a DT9840 Series module by name, sets the size the module input and output buffers, and returns a handle to the module. This handle is then used by all subsequent communication functions to identify the module.

Parameters

Name: BoardName

Description: A fixed length STRING variable that specifies the ASCII name of the DT9840 Series module to open. You assign this name when you configure the device driver. If *BoardName* is a NULL string or empty string, the first DT9840 Series module that is connected to the host computer is opened. This is the simplest method of opening a module in systems where only one DT9840 Series module is installed.

Refer to the *DT9840 Series Getting Started Manual* for more information on assigning a module name.

Name: hBoard

Description: A variable of type LONG in which the handle to the open DT9840 Series module is returned. If the module cannot be opened, NULL is returned.

Name: BoardOptions

Description: A variable of type ANY. If you do not want to specify the size of the module input and output buffers, specify NULL for *BoardOptions* to set the module input and output buffers to their default sizes (64 kBytes). This is the normal operating mode.

If you want to specify the size of the module input and output buffers, specify a variable type of BOARD_OPTIONS, described on page 188.

Notes If you are using multiple modules and do not know their names, you can use **DT_BoardGetName**, described on page 88, to return the module names.

When you are finished with the module, call **DT_BoardClose**, described on page 85, to close the module and release any resources that were allocated to it.

Return Values See Appendix A starting on page 107.

Example

```
'Open the module DT9841(00),
'create internal buffers using the default
'buffer size, and return the handle in
'BoardHandle
Dim Status As DTSTATUS
Dim BoardHandle As Long

Status = DT_BoardOpen("DT9841(00)",
   BoardHandle, NULL)
```

DT_BoardReadFromMemory

Syntax

```
Declare Function DT_BoardReadFromMemory
   Lib "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByVal SrcDSPAddress As Long, ByVal NumBytes
   As Long,
   ByRef DestPCBuffer As Long
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Returns data from a DSP address on the DT9840 Series module to a variable on the host computer (with no assistance from the DSP).

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: SrcDSPAddress

Description: A variable of type LONG that specifies the DSP address on the DT9840 Series module that you want to read.

Name: NumBytes

Description: A variable of type LONG that specifies the number of bytes to read from the DSP on the DT9840 Series module.

Name: DestPCBuffer

Description: A variable of type LONG in which the data from the DSP on the DT9840 Series module is returned to the host.

Notes

Although you can read from any DSP address using this function, it is highly recommended that you read from a known DSP address. Reading from an arbitrary DSP address can cause major system problems.

This function operates synchronously and take precedence over all other communication operations. For example, if you use this function to read 2 MBytes of data from the DT9840 Series module, all other messaging between the host computer and the module is suspended until all 2 MBytes has been transferred.

Unlike most other functions in this library, **DT_BoardReadFromMemory** does not require that a DSP program is running on the module.

To read specific registers on the DT9840 Series module, use **DT_BoardReadRegister**, described on page 91.

Return Values

See Appendix A starting on page 107.

Example

```
Dim Status As DTSTATUS
Dim DSPAdd As Long
Dim HostValue As Long
'Read 4 bytes from the memory location
'specified by DSPAdd and store it in HostValue

Status = DT_BoardReadFromMemory(BoardHandle,
   DSPAdd, 4, HostValue)
```

DT_BoardReadRegister

Syntax

```
Declare Function DT_BoardReadRegister Lib
   "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByVal Register As Long,
   ByRef Value As Long
   ) As DTSTATUS
```

Public Interface

DTCommLib.BAS

Description Reads the value of a specific register on a DT9840 Series module.

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: Register

Description: A variable of type LONG that specifies the register on the DT9840 Series module to read.

Name: Value

Description: A variable of type LONG in which the value of the specified register on the DT9840 Series module is returned.

Notes The DTCommAPI.BAS file defines constants that you can use for *Register*. Refer to page 164 for more information on these constants.

Return Values See Appendix A starting on page 107.

Example

```
'Read the raw value of the register for analog
'input channel 0 and store it in Value
Dim Status As DTSTATUS
Dim BoardHandle As Long
Dim Value As Long

Status = DT_BoardReadRegister(BoardHandle,
   AD0_IN_REG, Value)
```

DT_BoardRun

Syntax

```
Declare Function DT_BoardRun
   Lib "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByRef Args As String
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Runs a DSP program that was downloaded previously using **DT_BoardDownload**, described on page 86.

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: Args

Description: Support for this argument is currently not implemented; specify NULL for *Args*.

A fixed length STRING variable that contains new arguments for the DSP program. This string is parsed and passed to the main procedure of the DSP program. Specify NULL for *Args* if you have no new arguments for the DSP program.

Notes When a DSP program is downloaded (using **DT_BoardDownload**, described on page 86), the program is loaded into DSP memory but is not started. **DT_BoardRun** starts program execution.

If a DSP program is already running when this function is called, **DT_BoardRun** resets the DSP, passes new arguments to the program, and re-runs the program.

Return Values See Appendix A starting on page 107.

Example

```
'On the module specified by BoardHandle, run
'the previously downloaded program, changing
'none of the program arguments.
Dim Status As DTSTATUS
Dim BoardHandle As Long
Status = DT_BoardRun(BoardHandle, 0)
```

DT_BoardWriteRegister

Syntax

```
Declare Function DT_BoardWriteRegister
   Lib "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByVal Register As Long,
   ByVal Value As Long
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Writes a value to a specified register on the DT9840 Series module.

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: Register

Description: A variable of type LONG that specifies the register on the DT9840 Series module to write to.

Name: Value

Description: A variable of type LONG that specifies the value to write to the specified register on the DT9840 Series module.

Notes The DTCommAPI.BAS file defines constants that you can use for *Register*. Refer to page 164 for more information on these constants.

Return Values See Appendix A starting on page 107.

Example

```
'On the module specified by BoardHandle, set
'the state of the debug LEDs
Dim Status As DTSTATUS
Dim BoardHandle As Long
Dim CurrentLEDValue Mask, i As Long
CurrentLEDValue = 0
Mask = 1
For i = 0 to 7
  If LED(i).Value = 1 Then
   CurrentLEDValue = CurrentLEDValue or Mask
  End If
  Mask = Mask * 2
Next

Status = DTBoardWriteRegister(BoardHandle,
   LED_REG,CurrentLEDValue)
```

DT_BoardWriteToMemory

Syntax

```
Declare Function DT_BoardWriteToMemory
   Lib "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByRef SrcPCBuffer As Long,
   ByVal NumBytes As Long,
   ByVal DestDSPAddress As Long
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Writes data from a variable on the host computer to an address in DSP memory on the DT9840 Series module (with no assistance from the DSP).

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: SrcPCBuffer

Description: A variable of type LONG that contains the data from the host to write to the DSP on the DT9840 Series module.

Name: NumBytes

Description: A variable of type LONG that specifies the number of bytes to write to the DSP on the DT9840 Series module.

Name: DestDSPAddress

Description: A variable of type LONG that specifies the address of DSP memory on the DT9840 Series module to write to.

Notes

Although you can write to any DSP address using this function, it is highly recommended that you write to a known DSP address. Writing to an arbitrary DSP address can cause major system problems.

This function operates synchronously and take precedence over all other communication operations. For example, if you use this function to write 2 MBytes of data from the host computer to the DSP of the module, all other messaging between the host computer and the module is suspended until all 2 MBytes has been transferred.

Unlike most other functions in this library, **DT_BoardWriteToMemory** does not require that a DSP program is running on the module.

To write to specific registers on the DT9840 Series module, use **DT_BoardWriteRegister**, described on page 93.

Return Values

See Appendix A starting on page 107.

Example

```
Dim Status As DTSTATUS
Dim HostValue As Long
Dim DSPAdd As Long
'Write the contents of HostValue to the memory
'location specified by DSPAdd
Status = DT_BoardWriteToMemory(BoardHandle,
   HostValue, 4, DSPAdd)
```

DT_ChanAsyncWrite

Syntax

```
Declare Function DT_ChanAsyncWrite Lib
   "DTCommLib.dll" (
   ByVal hChan As Long,
   ByRef Buffer As Any,
   ByVal NumBytes As Long
   ) As DTSTATUS
```

Public Interface

DTCommLib.BAS

Description Asynchronously writes a specified number of bytes from an open communication channel on the host to the DT9840 Series module.

Parameters

Name: hChan

Description: A variable of type LONG that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 99.

Name: Buffer

Description: A variable of type ANY that contains the data to write to the DT9840 Series module. You can write into any data type, structure, or array.

Name: NumBytes

Description: A variable of type LONG that specifies the number of bytes to write to the DT9840 Series module.

Notes This function writes the data and returns immediately. It does not wait for an acknowledgement from the module. If the communication channel is busy or backed up with data, the actual transfer to the module continues in the background; this function does not block.

Return Values See Appendix A starting on page 107.

Example

```
'Asynchronously write the contents of the 2
'Byte variable "Value1" to "ChannelHandle",
'which was previously opened as WRITE_CHAN
'on the host.
Dim Status As DTSTATUS
Dim ChannelHandle As Long
Dim Value1 As Integer
Dim NumBytes As Long
Value1 = 255
NumBytes = 2

Status = DT_ChanAsyncWrite(ChannelHandle,
   Value1, NumBytes)
```

DT_ChanBytesAvailable

Syntax

```
Declare Function DT_ChanBytesAvailable
   Lib "DTCommLib.dll" (
   ByVal hChan As Long,
   ByRef BytesAvailable As Long
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Returns the number of bytes that are currently available in the specified channel buffer.

Parameters

Name: hChan

Description: A variable of type LONG that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 99.

Name: BytesAvailable

Description: A variable of type LONG in which the number of bytes is returned.

Return Values See Appendix A starting on page 107.

Example

```
'Read the number of bytes available in the
'channel specified by the handle hChan
'and store it in AvailableBytes
Dim Status As DTSTATUS
Dim hChan As Long
Dim AvailableBytes As Long

Status = DT_ChanBytesAvailable(hChan,
   AvailableBytes)
```

DT_ChanClose

Syntax

```
Declare Function DT_ChanClose
   Lib "DTCommLib.dll" (
   ByVal hChan As Long
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Closes a previously opened communication channel on the host and releases all allocated resources that were associated with it.

Parameters

Name: hChan

Description: A variable of type LONG that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 99.

Notes None

Return Values See Appendix A starting on page 107.

Example

```
'Close channel ChannelHandle on the host
Dim Status As DTSTATUS

Status = DT_ChanClose(ChannelHandle)
```

DT_ChanGetInfo

Syntax

```
Declare Function DT_ChanGetInfo Lib
   "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByVal ChanIndex As Integer,
   ByVal ChanName As String,
   ByRef ChanType As CHAN_TYPE,
   ByRef HostBufSize As Long,
   ByRef DSPBufSize As Long
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Returns information about the channel that is associated with the specified *ChanIndex*.

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: ChanIndex

Description: A variable of type INTEGER that specifies the index of the communication channel. The first open communication channel has a *ChanIndex* of 0.

Name: ChanName

Description: A fixed length STRING variable in which the ASCII name of the active communication channel is returned.

Name: ChanType

Description: A variable of CHAN_TYPE, described on page 171, in which the type of operation for the communication channel is returned.

Name: HostBufSize

Description: A variable of type LONG in which the size of the channel input buffer on the host computer is returned.

Name: DSPBufSize

Description: A variable of type LONG in which the size of the channel input buffer on the DT9840 Series module is returned.

Notes

This function is rarely needed. It is used only in very generalized cases, where for example, an application requires the names of all active channels at runtime.

More typical programs deal with specific channels with well known names. In these cases, the application can open the channel directly by name using **DT_ChanOpen**, described on page 99.

Return Values

See Appendix A starting on page 107.

Example

```
'Get the information for the first
'communication channel on the host and print
'it to the immediate/debug window
Dim Status As DTSTATUS
Dim BoardHandle As Long
Dim ChanName As String * 200
Dim ChanType As CHAN_TYPE
Dim HostBufSize As Long
Dim DSPBufSize As Long
Status = DT_ChanGetInfo(BoardHandle, 0,
   ChanName, ChanType, HostBufSize,
   DSPBufSize)

  Debug.Print "Index = 0"
  Debug.Print "Channel Name "; ChanName
  Debug.Print "Host Buffer Size "; HostBufSize
  Debug.Print "DSP Buffer Size "; DSPBufSize
  If ChanType = READ_CHAN Then Debug.Print
   "Channel Type:| READ_CHAN"
  Else
   Debug.Print "Channel Type:WRITE_CHAN"
  End If
```

DT_ChanOpen

Syntax

```
Declare Function DT_ChanOpen
   Lib "DTCommLib.dll" (
   ByVal hBoard As Long,
   ByVal ChanName As String,
   ByVal ChanType As CHAN_TYPE,
   ByVal BufSize As Long,
   ByRef hChan As Long,
   ByVal TimeoutInMS As Long
   ) As DTSTATUS
```

Public Interface

DTCommLib.BAS

Description Opens a unidirectional stream-based communication channel by name, creates an input buffer for the channel if it was opened for reading, and returns a handle to that channel. This handle is then used by all subsequent channel communication functions to identify the channel.

Parameters

Name: hBoard

Description: A variable of type LONG that specifies the handle to the opened DT9840 Series module. This value is returned by **DT_BoardOpen**, described on page 89.

Name: ChanName

Description: A fixed length, NULL-terminated STRING variable that specifies the name of the communication channel to open.

Name: ChanType

Description: A variable of type CHAN_TYPE, described on page 171, that specifies the type of operation to perform on the communication channel.

An unsigned integer variable or constant that specifies the size, in bytes, of the input buffer for a communication channel that was opened for reading.

If *ChanType* = READ_CHAN, *BufSize* determines how much data the communication channel on the host computer can receive from the DT9840 Series module at one time. In general, the size of the channel input buffer should be at least as large as the largest single write of data from the module. Note that a channel input buffer that is larger than needed has no negative performance impact on I/O operations.

If *ChanType* = WRITE_CHAN, *BufSize* is ignored on the host, but is passed to the module. Therefore, you could use this value to indicate to the DSP program how large to create the matching channel input buffer on the module, if you wish.

Name: hChan

Description: A variable of type LONG in which the handle to the opened communication channel is returned. If this procedure fails, *hChan* is NULL.

Name: TimeoutInMS

Description: A variable of type LONG that specifies the timeout value, in milliseconds. If *TimeoutInMS* = 0, the timeout period is disabled.

Notes

Once the communication channel is open, you can use the other channel functions to send data to, receive data from, and manage the communication channel.

A host application can open an arbitrarily named channel and write data to or read data from it. The corresponding program on the DT9840 Series module can then use this name to open the channel and establish communication with it.

The DT9840 Series module must open its side of the channel within the specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application to detect the timeout error and recover, if possible.

When dealing with multiple channels, ensure that your host and DSP applications open the communication channels in the same order. For example, if your host application opens channels 1, 2 and 3, ensure that your DSP program opens channel 1 before it opens channel 2, and opens channels 1 and 2 before opening channel 3.

When you are finished with it, ensure that you close the communication channel using **DT_ChanClose**, described on page 97, to release any resources that were associated with it.

Return Values

See Appendix A starting on page 107.

Example

```
'Open "CHAN1" on the host for writing and
'return the handle to the channel in
'ChannelHandle. If the DSP program does not
'open its side of the channel in 10000 ms,
'time out.
Dim Status As DTSTATUS
Dim BoardHandle As Long
Dim ChannelHandle As Long
Status = DT_ChanOpen(BoardHandle, “CHAN1”,
   WRITE_CHAN, 0, ChannelHandle, 10000)
```

DT_ChanRead

Syntax

```
Declare Function DT_ChanRead
   Lib "DTCommLib.dll" (
   ByVal hChan As Long,
   ByRef Buffer As Any,
   ByVal NumBytes As Long,
   ByVal TimeoutInMS As Long
   ) As DTSTATUS
```

Public Interface

DTCommLib.BAS

Description

Returns a specified number of bytes from a previously opened communication channel.

Parameters

Name: hChan

Description: A variable of type LONG that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 99.

Name: Buffer

Description: A variable of type ANY in which the data from the DT9840 Series module is returned. You can read from any data type, structure, or array.

Name: NumBytes

Description: A variable of type LONG that specifies the number of bytes to read from the DT9840 Series module.

Name: TimeoutInMS

Description: A variable of type LONG that specifies the timeout value, in milliseconds. If *TimeoutInMS* = 0, the timeout period is disabled.

Notes

If the requested number of bytes is not yet available for the channel, this function blocks all operations until either the data is available or the function times out. If you prefer a non-blocking operation, use **DT_ChanReadAvailable**, described on page 103.

When the host application calls **DT_ChanRead**, the DSP application must write *NumBytes* bytes of data to the communication channel within the specified timeout period, or a DT_TIMEOUT error is returned. You can write the host application to detect the timeout error and recover, if possible.

Return Values

See Appendix A starting on page 107.

Example

```
'Read 4 bytes from ChannelHandle,
'and put it in Value1. If the module does not
'write the specified number of bytes in
'1000 ms, a timeout occurs.
Dim Status As DTSTATUS
Dim ChannelHandle As Long
Dim Value1 As Single

Status = DT_ChanRead(ChannelHandle, Value1, 4,
   1000)
```

DT_ChanReadAvailable

Syntax

```
Declare Function DT_ChanReadAvailable Lib
   "DTCommLib.dll" (
   ByVal hChan As Long,
   ByRef Buffer As Any,
   ByVal BufSize As Long,
   ByRef BytesRead As Long
   ) As DTSTATUS
```

Public Interface DTCommLib.BAS

Description Returns the data that is currently available from an open communication channel on the DT9840 Series module, up to the size of the channel input buffer (*BufSize*).

Parameters

Name: hChan

Description: A variable of type LONG that specifies the handle to the open communication channel. This value is returned by **DT_ChanOpen**, described on page 99.

Name: Buffer

Description: A variable of type ANY in which the data from the DT9840 Series module is returned. You can read from any data type, structure, or array.

Name: BufSize

Description: A variable of type LONG that specifies the size of the *Buffer* variable, in bytes. This value is the maximum number of bytes that can be read from the DT9840 Series module and stored in the *Buffer* variable at one time.

Name: BytesRead

Description: A variable of type LONG in which the number of bytes that were read is returned; this value can never be greater than *BufSize*.

When *the number that is returned* is less than *BufSize*, not all of the data from the DT9840 Series module is available; but, the data that is available is returned to the host application.

When *the number that is returned* is equal to *BufSize*, more than the requested data from the DT9840 Series module is available, but only the portion of data that fits in the *Buffer* variable is returned.

Notes The most data that can be returned at one time cannot be larger than the size specified by *BufSize*.

When this function is called, the available data, up to the size of the *Buffer* variable, is returned. If no data is available, other operations can continue.

If you want to block other operations from continuing while you are waiting for all the data to become available, use the **DT_ChanRead** function, described on page 101, instead.

Return Values See Appendix A starting on page 107.

Example

```
'Read the number of bytes available from
'"ChannelHandle" up to (20 bytes), and put it
'in Value1. Return the actual number of bytes
'read in BytesRead.ChannelHandle" was
'previously opened as READ_CHAN on the host.
Dim Status As DTSTATUS
Dim ChannelHandle As Long
Dim Value1 As Single
Dim BytesRead As Long

Status = DT_ChanReadAvailable(ChannelHandle,
   Value1, 20, BytesRead)
```

DT_GetErrorString

Syntax

```
Declare Sub DT_GetErrorString
   Lib "DTCommLib.dll" (
   ByVal Status As DTSTATUS,
   ByVal EString As String)
```

Public Interface DTCommLib.BAS

Description Returns an ASCII string that corresponds to a specific error.

Parameters

Name: Status

Description: A variable of type DSTATUS, described on page 178, that specifies the error for which an ASCII string descriptor is returned.

Name: EString

Description: A fixed length STRING variable in which the error descriptor is returned.

Notes All of the functions in the DT9840 Series DSP Library return errors that you can use with this function.

Return Values None

Example

```
'Utility for reporting DTSTATUS errors. The
'error description for Status is returned in
'ErrString.
Dim Status As DTSTATUS
Dim ErrString As String * 256

If (Status <> DT_SUCCESS) Then
  Call DT_GetErrorString(Status, ErrString)
End If
```

DT_VoltsToDAScan

Syntax

```
Declare Function DT_VoltsToDAScan Lib
   "DTCommLib.dll" (
   ByRef Volts As Double,
   ByRef OutScan As
     OUTPUT_SCAN_RCD,
   ByVal VoltageRange As DAC_RANGE
   ) As DTSTATUS
```

Public Interface

DTCommLib.BAS

Description

Converts an array of voltage values into a scan of raw analog output values for use with **DT_WriteDACDIO**, **DT_ScanLoop**, or **DT_FunctionGen**.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_WriteDACDIO**, **DT_ScanLoop**, or **DT_FunctionGen** functions.

Parameters

Name: Volts

Description: A variable of type DOUBLE that contains the voltage values to convert into raw analog output values.

Allocate *Volts* to hold NUM_DAC_CHANS voltage values; for the DT9841, DT9841E, DT9841-VIB, and DT9842/2, the value of NUM_DAC_CHANS is 2. For the DT9842/8, the value of NUM_DAC_CHANS is 8.

Name: OutScan

Description: A variable of type OUTPUT_SCAN_RCD, described on page 205, in which the raw analog output values are returned.

Name: VoltageRange

Description: A variable of type DAC_RANGE, described on page 177, that specifies the analog output voltage range.

Notes

Avoid this function during time-critical operations, due to the overhead required.

Return Values

See Appendix A starting on page 107.

Example

```
'Convert the voltage values in the array
'OutVolts, which are in the +/- 10 V range,
'into raw values and store them in OutScans.
Dim OutVolts(NUM_DAC_CHANS - 1) As Double
Dim OutScans As OUTPUT_SCAN_RCD

Status = DT_VoltsToDAScan( OutVolts(0),
   OutScans, BIP10_VOLTS_SPAN)
```

DT_VoltsToDAValue

Syntax

```
Declare Function DT_VoltsToDAValue Lib
   "DTCommLib.dll" (
   ByVal Volts As Double,
   ByRef DAValue As Long,
   ByVal VoltageRange As DAC_RANGE
   ) As DTSTATUS
```

Public Interface

DTCommLib.BAS

Description

Converts a single voltage value into a raw analog output value that is used by **DT_WriteDAC**.

Refer to the *DT9840 Series DSP Library User's Manual* for more information on the **DT_WriteDAC** function.

Parameters

Name: Volts

Description: A variable of type DOUBLE that specifies the voltage value to convert into a raw analog output value.

Name: DAValue

Description: A variable of type LONG in which the converted raw analog output value is returned.

Name: VoltageRange

Description: A variable of type DAC_RANGE, described on page 177, that specifies the analog output voltage range.

Notes

Avoid this function during time-critical operations, due to the overhead required.

Return Values

See Appendix A starting on page 107.

Example

```
'Convert the voltage in Volts, which is in the
' +/- 10 V range, into a raw value and store
'in Value
DIM Status As DTSTATUS
Dim Volts As Double
Dim Value As Long
Status = DT_VoltsToDAValue(Volts, Value,
   BIP10_VOLTS_SPAN)
```

Error Codes

Table 4 lists the error codes that can be returned or detected by the DT9840 Series Host Communication Library.

Table 4: Errors that the DT9840 Series Host Communication Library Can Return or Detect

Error Code	Error Message	ASCII Text	Solution
0	DT_SUCCESS	Success	No error; the function executed successfully.
1	DT_NO_MEMORY	Not enough memory available to create internal buffer.	Request smaller buffers or increase the amount of system memory.
2	DT_BUFFER_TOO_SMALL	Incoming message is too big for caller's buffer.	Increase the size of the buffer so that it is as least as large as the largest message that you will receive from the DT9840 Series module.
3	DT_INVALID_CHAN_NAME	The *pChanName* argument is not valid.	Check the channel name and ensure that the specified channel exists.
4	DT_INVALID_CHAN_HANDLE	The channel handle argument is not valid.	Check that the channel handle is a valid handle that was returned by **DT_ChanOpen**.
5	DT_INVALID_PHCHAN	The *phChan* argument is not valid.	Pass a valid pointer.
6	DT_NOT_IMPLEMENTED	This function is not yet implemented.	Do not call this function at this time.
7	DT_INVALID_CALLBACK_PTR	The *pCallback* argument is not valid pointer.	Pass a valid pointer for *pCallback*.
8	DT_INVALID_BUFFER_PTR	The *pBuffer* argument is not valid.	Pass a valid pointer for *pBuffer*.
9	DT_INVALID_NUM_BYTES	The *NumBytes* argument is not valid.	Specify a value for *NumBytes* that is great0er than 0. If you are using **DT_BoardWriteTo Memory**, also ensure that *NumBytes* is divisible by 4.
10	DT_INVALID_BYTES_READ_ PTR	The *pBytesRead* argument is an invalid pointer.	Pass a valid pointer for *pBytesRead*.
11	DT_TIMEOUT	The operation timed out.	Ensure that the DSP application responds to the host within the specified timeout period.
12	DT_INVALID_CHAN_TYPE	The *ChanType* argument is not valid.	Specify READ_CHAN to read data from the channel or WRITE_CHAN to write data to the channel.
13	DT_INVALID_POINTER	An invalid pointer was passed as an argument.	Pass a valid pointer to an appropriate variable.
14	DT_CALLBACK_NOT_FOUND	An attempt was made to unregister a callback that wasn't registered.	Check the name of the callback function to unregister. You can unregister callback functions only if they have been registered.

Table 4: Errors that the DT9840 Series Host Communication Library Can Return or Detect (cont.)

Error Code	Error Message	ASCII Text	Solution
15	DT_CHAN_DISCONNECTED	Remote end of communication channel has disconnected.	Reconnect the USB cable between the host computer and the DT9840 Series module.
16	DT_INCOMPATIBLE_CHAN_TYPE	Remote end of channel was opened as incompatible type (R vs W).	If the host application opened the communication channel for a write operation, open the communication channel on the module for a read operation. If the host application opened the communication channel for a read operation, open the communication channel on the module for a write operation.
17	DT_CHAN_ALREADY_OPEN	The specified channel is already open.	The specified communication channel is already open. Check the channel name. You need only open a communication channel once in both the host application and DSP program.
18	DT_CHAN_ALREADY_CLOSED	The specified channel is already closed.	The specified communication channel is already closed. Check the channel name. You need only close a communication channel once in both the host application and DSP program.
19	DT_TOO_BIG_FOR_CHAN_BUFFER	The requested size of write is bigger than receiver's chan buffer.	Ensure that the channel input buffer on the module is large enough to hold the data that is being written by the host.
20	DT_TOO_BIG_FOR_BOARD_BUFFER	The requested size of a write is bigger than board's send buffer.	Ensure that the module input buffer on the host is large enough to hold the data that is being written from the module.
21	DT_VOLTAGE_OUT_OF_RANGE	Voltage being converted is out of range of D/A.	The voltage value must be within +/-10 V.
22	DT_AD_VALUE_OUT_OF_RANGE	A/D value is out of range (+ or - full scale).	The A/D value must be 32 bits wide and left justified; the right eight bits must be filled with zeros.
23	DT_NO_SYNC_DATA_RECEIVED	No synchronous msg was received but Acknowledge was sent.	An acknowledge should be sent only when the host receives a synchronous write command from the module.
24	DT_INVALID_DSP_BUFFER_SIZE_PTR	An invalid pointer was passed as the *pDSPBufSize* argument.	Pass a valid pointer for *pDSPBufSize*.
25	DT_INVALID_HOST_BUFFER_SIZE_PTR	An invalid pointer was passed as the *pHostBufSize* argument.	Pass a valid pointer for *pHostBufSize*.

Table 4: Errors that the DT9840 Series Host Communication Library Can Return or Detect (cont.)

Error Code	Error Message	ASCII Text	Solution
26	DT_INVALID_BUF_SIZE	The *BufSize* argument is invalid.	Specify a value for *BufSize* that is greater than 0. In general, the value of *BufSize* should be at least as large as the largest single write of data from the DSP application.
27	DT_CHAN_OPENED_FOR_WRITE	Tried to read from a channel that was opened for writing.	Open the channel as type READ_CHAN with **DT_ChanOpen.**
28	DT_CHAN_OPENED_FOR_READ	Tried to write to a channel that was opened for reading.	Open the channel as type WRITE_CHAN with **DT_ChanOpen.**
1000	DT_INVALID_BOARD_NAME	The *BoardName* argument is invalid.	Check the module name using **DT_BoardGetName**.
1001	DT_INVALID_BOARD_HANDLE	The board handle argument is not valid.	Specify an appropriate module handle that was returned by **DT_BoardOpen**.
1002	DT_INVALID_PHBOARD	The *phBoard* argument is not valid.	Pass a valid pointer for *phBoard*.
1003	DT_INVALID_FILENAME	The *pFilename* argument is not valid.	Pass a valid pointer for *pFilename*.
1004	DT_INVALID_BOARD_INDEX	The *BoardIndex* argument is not valid.	If a single module is installed in the system, specify 0 for the board index. If more than one module is installed in the system, specify an appropriate value for the board index. The modules are numbered sequentially starting at 0.
1005	DT_INVALID_BOARD_INFO	The *pBoardInfo* argument is not valid.	Pass a valid pointer for *pBoardInfo*.
1006	DT_INVALID_WINDOW_HANDLE	The *hWindow* argument is not valid.	Specify an appropriate window handle.
1007	DT_FAILURE	General Error, indicates an internal error.	This error indicates that an internal problem occurred in the DT9840 Series Host Communication Library. This error should never occur in a properly installed and running system. Contact Technical Support; see page 17.
1008	DT_INVALID_MSG_PTR	The *pMsg* argument is not valid.	Pass a valid pointer for *pMsg*.
1009	DT_INVALID_MSG_BUF_LEN	The *MsgBufLen* argument is not valid.	This error indicates an internal communication error. This error should never occur in a properly installed and running system. Contact Technical Support; see page 17.

Table 4: Errors that the DT9840 Series Host Communication Library Can Return or Detect (cont.)

Error Code	Error Message	ASCII Text	Solution
1010	DT_INVALID_DSP_ADDRESS	The *DSPAddress* must be an even multiple of 4.	Specify an appropriate value for *DSPAddress*. This value must be an even multiple of 4. Refer to the *DT9840 Series User's Manual* for DSP addresses.
1011	DT_CANT_OPEN_DRIVER	Cannot open the DT9841 device driver.	This error should never occur in a properly installed and running system. Typically, this indicates that a software installation problem occurred. Contact Technical Support; see page 17.
1012	DT_CANT_FIND_BOARD	Cannot find a board with the specified name.	Check the module name using **DT_BoardGetName**.
1013	DT_NO_BOARD_FOR_INDEX	No board exists for specified board index.	If a single module is installed in the system, specify 0 for the board index. If more than one module is installed in the system, specify an appropriate value for the board index. The modules are numbered sequentially starting at 0.
1014	DT_NO_CHAN_FOR_INDEX	No channel exists for specified channel index.	Check the board index. If the index exists, open a communication channel for that module with **DT_ChanOpen**.
1015	DT_NO_MESSAGING	Messaging system is not yet running.	This error should never occur in a a properly installed and running system. It indicates that the messaging system was not started properly when the module was opened with **DT_BoardOpen**. Contact Technical Support; see page 17.
1016	DT_WINDOW_NOT_FOUND	An attempt was made to unregister a window that wasn't registered.	Check the name of the message handler to unregister. You can unregister message handlers functions only if they have been registered.
1017	DT_FILE_NOT_FOUND	The specified file does not exist.	Specify an appropriate filename for the Code Composer (COFF) file.
1018	DT_DRIVER_VERSION	The currently installed Windows driver is the wrong version.	This version of the library is not compatible with the DT9840 Series Device Driver that is installed on the host computer. This error should never occur in a properly installed and running system. Typically, this error indicates that a software installation problem occurred with the module. Contact Technical Support; see page 17.

Table 4: Errors that the DT9840 Series Host Communication Library Can Return or Detect (cont.)

Error Code	Error Message	ASCII Text	Solution
1019	DT_FIRMWARE_VERSION	Version of DT9840Ld.sys driver not compatible with dt9840k.sys	This version of the DT9840 Series Communication Library is not compatible with the DT9840 Series module that you are attempting to communicate with. This error should never occur in a properly installed and running system. It indicates a compatibility problem between the host software and the software on the module. Contact Technical Support; see page 17.
1020	DT_DSP_VERSION	The versions of the DSP program and the DT9841 are incompatible.	The version of the DSP application program that was downloaded to the DT9840 Series module is incompatible with the USB communication software on the module. This error should never occur in a properly installed and running system. Contact Technical Support; see page 17.
1021	DT_NO_PROGRAM_LOADED	Can't run program because none have been downloaded yet.	Download a program using **DT_BoardDownload** before calling **DT_BoardRun**.
1022	DT_ERROR_FILE_READ	Error in reading from file.	This error indicates that an internal problem occurred while reading a COFF file. The most likely cause is that the file is corrupted. This error should never occur when loading a valid COFF file. Check your file.
1023	DT_ERROR_INVALID_COFF	File to be downloaded is not COFF formatted.	Specify a file that adheres to the Code Composer format (COFF).
1024	DT_INVALID_BUFFER_ ADDRESS	The *DSPAddress* argument does not reference a valid buffer.	Specify a valid DSP address. Refer to the *DT9840 Series User's Manual* for DSP addresses.
1025	DT_ERROR_MEMORY_ MISMATCH	Mismatch in value written to and read back from memory.	A COFF file was downloaded to the DT9840 Series, and the DSP memory was verified against the original memory image on the host. At least one byte was incorrect. This indicates that either the COFF file attempted to load data at addresses that do not refer to memory on the DT9840 Series or a hardware/communication problem exists with the module. Contact Technical Support; see page 17.
1026	DT_USB_ERROR	An unknown type of USB error occurred, indicates a system problem.	This error should never occur in a properly installed and running system. Contact Technical Support; see page 17.

Table 4: Errors that the DT9840 Series Host Communication Library Can Return or Detect (cont.)

Error Code	Error Message	ASCII Text	Solution
1027	DT_INVALID_DSP_ADDRESS_PTR	An invalid pointer was passed as the *pDSPAddress* argument.	Pass a valid pointer for *pDSPAddress*.
1028	DT_INVALID_REGISTER	An invalid address was passed as the Register argument.	Specify the address of the appropriate register. Refer to the *DT9840 Series User's Manual* for more information on available registers.
1029	DT_BOARD_ALREADY_OPEN	Board is already opened by another program.	Check the module name. Specify a name for a module that is not already in use.
1030	DT_INVALID_BOARD_OPTIONS	BoardOptions structure contains illegal values.	Check the values for the BoardOptions structure, described on page 141 for Visual C++ users and on page 188 for Visual Basic users.
1031	DT_BOARD_NOT_RESPONDING	DT9841 is not responding	Check that the USB cable is still plugged into the module and that the power switch of the module is turned on. If you still get this error, contact Technical Support; see page 17.
1032	DT_BOARD_WAS_SHUTDOWN	DT9841 error occurred, board must be closed then reopened.	Close the module using **DT_BoardClose**, and then reopen the module using **DT_BoardOpen**.
1033	DT_PROGRAM_TOO_LARGE	Program is too large to be loaded into DT9840's flash memory.	This error is used internally by the DT9840 Series Flash Programmer Utility. It is returned to this utility when you attempt to download a program that is bigger than the available flash memory.
3000	DT_HOST_CHAN_ALREADY_OPEN	An OPEN_CHAN message received from DT9841 for an already open channel.	The specified communication channel is already open. Check the channel name. You need only open a communication channel once in both the DSP program and host application.
3001	DT_HOST_CHAN_ALREADY_CLOSED	A CLOSE_CHAN message received from DT9841 for a channel that's already closed.	Check the channel name. You need only close a communication channel once in both the DSP program and host application.
3002	DT_HOST_BUFFER_EMPTY	Host buffer is empty (and shouldn't be).	Ensure that the host application fills the output buffer in a timely manner.

Table 4: Errors that the DT9840 Series Host Communication Library Can Return or Detect (cont.)

Error Code	Error Message	ASCII Text	Solution
3003	DT_HOST_RCV_BUFFER_ FULL	Host's board's receive buffer is full.	To avoid this error, either increase the size of the module input buffer on the host by changing the *DSPToHostBufSizeInKb* parameter of the BOARD_OPTIONS structure that is passed to **DT_BoardOpen**, or read data more quickly from the host. If this problem still persists, ensure that all of the input channel buffers on the host are being read in a timely manner so that data does not back up.
3004	DT_HOST_MSG_UNKNOWN_ TYPE	Message received from DT9841 has an unknown message ID.	This error indicates that an internal communication error occurred. This error should never occur in a properly installed and running system. Contact Technical Support; see page 17.
3005	DT_HOST_MSG_INVALID_ LENGTH	Message received from DT9841 had invalid length.	This error indicates an internal communication error. This error should never occur in a properly installed and running system. Contact Technical Support; see page 17.
3006	DT_HOST_CHAN_BUFFER_ FULL	Host's channel buffer is full.	Either increase the size of the channel input buffer on the host (by calling **DT_ChanOpen** in the host program), or increase the rate at which the module reads data from the communication channel.
3007	DT_HOST_SEND_BUFFER_ FULL	Host's board's send buffer is full.	To avoid this error, either increase the size of the module output buffer on the host by changing the *HostToDSPBufSizeInKb* parameter of the BOARD_OPTIONS structure that is passed to **DT_BoardOpen**, or read data more quickly from the module. If this problem still persists, ensure that all of the channel buffers on the module are being read in a timely manner so that data does not back up.

Data Types, Constants, Enumerated Types, Structures, and Callback Functions

For Visual C++ Programmers 116
For Visual Basic Programmers 164

For Visual C++ Programmers

This section describes the data types, constants, enumerated types, structures, and callback functions that are used when calling the DT9840 Series host communication functions in Visual C++. Refer to Chapter 4 starting on page 43 for more information.

Data Types

The following data types are defined in the DTCommLib.H file:

- LONG – A 32-bit integer.
- DOUBLE – A double, which is always 64-bits IEE format.
- BOARD_NAME_MAX_LEN – The maximum number of characters for the module name is 256.
- ERR_STRING_MAX_LEN – The maximum number of characters for the error string is 256.
- *BRD_HANDLE – A pointer of type VOID.
- *FIND_HANDLE – A pointer of type VOID.
- BOARD_NAME[BOARD_NAME_MAX_LEN] – An array of type CHAR that contains 256 elements.
- ERROR_STRING[ERR_STRING_MAX_LEN] – An array of type CHAR that contains 256 elements.
- MAX_BOARD_OPTION_SIZE – The maximum number of elements for the BOARD_OPTIONS structure is 200.

The following data types are defined in the DTCommonAPI.H file, which is referenced in the DTCommLib.H file:

- CHAN_NAME_MAX_LEN – The maximum number of characters for the channel name is 200.
- *CHAN_HANDLE – A pointer of type VOID.
- CHAN_NAME[CHAN_NAME_MAX_LEN] – An array of type CHAR that contains 200 elements.

Constants

The following constants are provided for Visual C++:

- General-purpose constants, described on this page
- Conversion constants, described on page 118
- Register constants, described on page 119
- Error constants, described on page 121
- Event constants, described on page 122
- Message constants, described on page 122

- Counter/timer constants, described on page 122
- Flash memory constants, described on page 123

General-Purpose Constants

The following constants are defined in the DTCommonAPI.H file, which is referenced in the DTCommLib.H file, for general use:

- MIN_SAMPLE_RATE – The minimum sample rate. For the DT9841, DT9841E, and DT9841-VIB, this is 200 Hz. For the DT9842/2 and DT9842/8, this value is 0 Hz.
- MAX_SAMPLE_RATE – The maximum sample rate. This is 100000 Hz for all DT9840 Series modules.
- DT_BOARD_TYPE – The specific DT9840 Series module; this value is 9841 to identify the DT9841 and DT9841-VIB, 98412 to identify the DT9841E module, 9842 to identify the DT9842/2, or 9848 to identify the DT9842/8.
- NUM_ADC_REGS – The number of analog input registers. All DT9840 Series modules have 8 analog input registers. For the DT9841E, the first two registers correspond to analog input channels 0 and 1; the data in the remaining registers should be ignored.
- NUM_ADC_CHANS – The number of analog input channels. For the DT9841, DT9841-VIB, DT9842/2, and DT9842/8, this value is 8. For the DT9841E module, the value of NUM_ADC_CHANS is 2.
- NUM_COUNTERS – The number of counter/timer channels; for all DT9840 Series modules, this value is 3.
- NUM_DIN – The number of 24-bit digital input lines; for all DT9840 Series modules, this value is 1. For the DT9841-VIB, only the first 16-bits are accessible; the 8-remaining bits should be ignored.
- NUM_DOUT – The number of 24-bit digital output lines; for all DT9840 Series modules, this value is 1. For the DT9841-VIB, only the first 16-bits are accessible; the 8-remaining bits should be ignored.
- NUM_ADC_STEPS – The number of steps for the A/D converter on the module. For the DT9841, DT9841E, and DT9841-VIB, this value ranges from 1 to 24; for the DT9842/2 and DT9842/8, this value ranges from 1 to 16.
- ADC_COUNTS_PER_STEPS – The number of count changes for each step of the A/D converter. For the DT9841, DT9841E, and DT9841-VIB, this value ranges from 1 to 8; for the DT9842/2 and DT9842/8, this value ranges from 1 to 16.
- NUM_DAC_CHANS – The number of analog output channels; for the DT9841, DT9841E, DT9841-VIB, and DT9842/2, this value is 2; for the DT9842/8, this value is 8.
- NUM_DAC_STEPS – The number of steps for the D/A converter on the module. For the DT9841, DT9841E, and DT9841-VIB, this value ranges from 1 to 24; for the DT9842/2 and DT9842/8, this value ranges from 1 to 16.
- DAC_COUNTS_PER_STEPS – The number of count changes for each step of the D/A converter. For the DT9841, DT9841E, and DT9841-VIB, this value ranges from 1 to 8; for the DT9842/2 and DT9842/8, this value ranges from 1 to 16.

- NUM_INPUTS_IN_RCD – The number of analog input registers (NUM_ADC_REGS), number of counter/timer channels (NUM_COUNTERS), and number of 24-bit digital input lines (NUM_DIN) in the input scan record for the specified module type. For example, for the DT9841, NUM_INPUTS_IN_RCD is 12 (8 analog input registers, 3 counter/timer channels, and 1 24-bit digital input port).
- NUM_OUTPUTS_IN_RCD – The number of analog output channels (NUM_DAC_CHANS) and number of 24-bit digital output lines (NUM_DOUT) in the output scan record for the specified module type. For example, for the DT9841, NUM_INPUTS_IN_RCD is 3 (2 analog output channels and 1 24-bit digital output port).
- NUM_INPUTS_USED – The number of inputs defined in the input scan record (NUM_INPUTS_IN_RCD).
- NUM_OUTPUTS_USED – The number of outputs defined in the output scan record (NUM_OUTPUTS_IN_RCD).

Conversion Constants

The following conversion constants are defined in the DTCommonAPI.H file, which is referenced in the DTCommLib.H file; these constants are provided if you choose to write your own conversion routines instead of using the conversions routines provided in the DT9840 Series Host Communication Library:

- BIP10_VOLTS_SPAN – An input or output range of ±10 V; this is 20L (LONG).
- BIP2PT5_VOLTS_SPAN – An input or output range of ±2.5 V; this is 5L (LONG).
- BIP10_COUNTS_PER_VOLT – The number of counts per volt for the ±10 V range. This is defined as follows: ((DOUBLE) 0x10000 * 0x10000 / BIP10_VOLTS_SPAN).
- BIP2PT5_COUNTS_PER_VOLT – The number of counts per volt for the ±2.5 V range. This is defined as follows: ((DOUBLE) 0x10000 * 0x10000 / BIP2PT5_VOLTS_SPAN).
- BIP10_VOLTS_PER_COUNT – The number of volts per count for the ±10 V range of the specified module. This is defined as follows: (1/BIP10_COUNTS_PER_VOLT).
- BIP2PT5_VOLTS_PER_COUNT – The number of volts per count for the ±2.5 V range of the specified module. This is defined as follows: (1/BIP2PT5_COUNTS_PER_VOLT).
- BIP10_VOLTS_PER_STEP – The number of volts per step for the ±10 V range of the specified module. This is defined as follows: (BIP10_VOLTS_PER_COUNT * COUNTS_PER_STEP).
- BIP2PT5_VOLTS_PER_STEP – The number of volts per step for the ±2.5 V range of the specified module. This is defined as follows: (BIP2PT5_VOLTS_PER_COUNT * COUNTS_PER_STEP).
- BIP10_MAX_VOLTS – The voltage that corresponds to the maximum count value for the ±10 V range of the specified module. This is defined as follows: (10.L – BIP10_VOLTS_PER_COUNT).
- BIP2PT5_MAX_VOLTS – The voltage that corresponds to the maximum count value for the ±2.5 V range of the specified module. This is defined as follows: (2.5.L – BIP2PT5_VOLTS_PER_COUNT).

- BIP10_MIN_VOLTS – The voltage that corresponds to the minimum count value for the ±10 V range of the specified module. For all DT9840 Series modules, this is defined as –10L.
- BIP2PT5_MIN_VOLTS – The voltage that corresponds to the minimum count value for the ±2.5 V range of the specified module. This range (–2.5.L) is supported by the DT9841, DT9841E, and DT9841-VIB modules.

Register Constants

The following register constants are defined in the DTCommonAPI.H file, which is referenced in the DTCommLib.H file. These constants are provided if you want to read the registers of a DT9840 Series module using **DT_BoardReadRegister**, described on page 55, or **DT_BoardReadFromMemory**, described onpage 54, or write to the registers of a DT9840 Series module using **DT_BoardWriteRegister**, described on page 57, or **DT_BoardWriteToMemory**, described on page 58:

Note: Refer to the *DT9840 Series User's Manual* for more information on these registers.

- DEVICE_REG_BASE – The Hardware Control and Status register of the DT9840 Series module (address 0xB0000000).
- HDW_CTRL_REG – The Hardware Control and Status register (DEVICE_REG_BASE).
- HDW_CALIBRATION_REG – A calibration register (DEVICE_REG_BASE + 0x0004).
- DT9842_CLOCK_REG – The clock divider register for the DT9842/2 and DT9842/8 modules (DEVICE_REG_BASE + 0x0008).
- USB_CALIBRATION_REG – A USB writable calibration register (DEVICE_REG_BASE + 0x000C).
- ID_REG – The model ID and FPGA register (DEVICE_REG_BASE + 0x1000).
- AD0_IN_REG – The data register for analog input channel 0 (DEVICE_REG_BASE + 0x4000).
- AD1_IN_REG – The data register for analog input channel 1 (DEVICE_REG_BASE + 0x4004).
- AD2_IN_REG – The data register for analog input channel 2 (DEVICE_REG_BASE + 0x4008). Ignore this register for DT9841E modules.
- AD3_IN_REG – The data register for analog input channel 3 (DEVICE_REG_BASE + 0x400C). Ignore this register for DT9841E modules.
- AD4_IN_REG – The data register for analog input channel 4 (DEVICE_REG_BASE + 0x4010). Ignore this register for DT9841E modules.
- AD5_IN_REG – The data register for analog input channel 5 (DEVICE_REG_BASE + 0x4014). Ignore this register for DT9841E modules.
- AD6_IN_REG – The data register for analog input channel 6 (DEVICE_REG_BASE + 0x4018). Ignore this register for DT9841E modules.

- AD7_IN_REG – The data register for analog input channel 7 (DEVICE_REG_BASE + 0x401C). Ignore this register for DT9841E modules.
- DIG_IN_REG – The input register for digital input ports 0, 1, and 2, and the flag bits (DEVICE_REG_BASE + 0x4020). Port 2 is not used for the DT9841-VIB module.
- CTR0_IN_REG – The input register for counter/timer 0 (DEVICE_REG_BASE + 0x4024).
- CTR1_IN_REG – The input register for counter/timer 1 (DEVICE_REG_BASE + 0x4028).
- CTR2_IN_REG – The input register for counter/timer 2 (DEVICE_REG_BASE + 0x402C).
- LED_REG – The register for the diagnostic LEDs (DEVICE_REG_BASE + 0x4030). The following constants are provided for controlling individual LEDs:
  - LED_D0 – LED CR9 (0x01).
  - LED_D1 – LED CR10 (0x02).
  - LED_D2 – LED CR11 (0x04).
  - LED_D3 – LED CR8 (0x08).
  - LED_D4 – LED CR14 (0x10).
  - LED_D5 – LED CR7 (0x20).
  - LED_D6 – LED CR12 (0x40).
  - LED_D7 – LED CR13 (0x80).
  - LED_ALL – LEDs CR7 to CR14 (0xFF).
  - LED_HBEAT – LED CR7 (0x20).
  - LED_RED – LED CR7 (0x20). This LED turns red.
  - LED_GREEN – LED CR14 (0x10). This LED turns green.
- DAC0_REG – The data register for analog output channel 0 (DEVICE_REG_BASE + 0x8000).
- DAC1_REG – The data register for analog output channel 1 (DEVICE_REG_BASE + 0x8004).
- DIG_OUT_REG – The digital output register (DEVICE_REG_BASE + 0x8008). Port 2 is not used for the DT9841-VIB module.
- CTR0_PERIOD_REG – The period register for counter/timer 0 (DEVICE_REG_BASE + 0xC000).
- CTR1_PERIOD_REG – The period register for counter/timer 1 (DEVICE_REG_BASE + 0xC004).
- CTR2_PERIOD_REG – The period register for counter/timer 2 (DEVICE_REG_BASE + 0xC008).
- CTR0_PULSE_REG – The pulse register for counter/timer 0 (DEVICE_REG_BASE + 0xC010).
- CTR1_PULSE_REG – The pulse register for counter/timer 1 (DEVICE_REG_BASE + 0xC014).
- CTR2_PULSE_REG – The pulse register for counter/timer 2 (DEVICE_REG_BASE + 0xC018).

- CTR0_CTRL_REG – The control register for counter/timer 0 (DEVICE_REG_BASE + 0xC020).
- CTR1_CTRL_REG – The control register for counter/timer 1 (DEVICE_REG_BASE + 0xC024).
- CTR2_CTRL_REG – The control register for counter/timer 2 (DEVICE_REG_BASE + 0xC028).
- CTR_STATUS_REG – The status register for the counter/timers (DEVICE_REG_BASE + 0xC030).
- DIG_IN_MASK_REG – The mask register for digital input port 0 (DEVICE_REG_BASE + 0x10000). You can program this register to generate an interrupt when any bit of port 0 changes state.
- DIG_PORT0_REG – The register for digital I/O port 0 (DEVICE_REG_BASE + 0x10004).
- DIG_PORT1_REG – The register for digital I/O port 1 (DEVICE_REG_BASE + 0x10008).
- DIG_PORT2_REG – The register for digital I/O port 2 (DEVICE_REG_BASE + 0x1000C). Port 2 is not used for the DT9841-VIB module.
- SB_CTRL_REG – The Control and Address (CE3) register for the Scalable Bus (DEVICE_REG_BASE + 0x14000).
- SB_XFER_CTRL_REG – The Scalable Bus Transfer Control register (DEVICE_REG_BASE + 0x14004).
- SB_XFER_STATUS_REG – The Scalable Bus Transfer Status register (DEVICE_REG_BASE + 0x14008).
- SB_REG_BASE – The Scalable Bus write register (0xA0000000).
- SB_OUT_REG – The Scalable Bus write register (SB_REG_BASE).
- SB_IN_REG – The Scalable Bus input register (SB_REG_BASE + 4).

Error Constants

The following error constants are defined in the DTErrors.H file, which is referenced by DTCommLib.H:

- DT_COMMON_BASE_ERR – A value of 0.
- DT_HOST_BASE_ERR – A value of 1000.
- DT_9841_BASE_ERR – A value of 2000.
- DT_HOST_COMM_BASE_ERR – A value of 3000.
- DT_9841_COMM_BASE_ERR – A value of 4000.
- DT_LAST_ERR_BASE – A value of 4000.

Event Constants

The following event constants are defined in the DTCommon.H file, which is referenced by DT9841.H:

- DTEVENT_REMOTE_CHAN_OPENED – A value of 0x00000001.
- DTEVENT_REMOTE_CHAN_CLOSED – A value of 0x00000002.
- DTEVENT_ASYNC_DATA_RECEIVED – A value of 0x00000004.
- DTEVENT_SYNC_DATA_RECEIVED – A value of 0x00000008.
- DTEVENT_CHAN_IN_BUFFER_FULL – A value of 0x00000010.
- DTEVENT_OUT_BUFFER_SPACE_AVAILABLE – A value of 0x00000020.
- DTEVENT_ERROR – A value of 0x00000040.
- DTEVENT_MODULE_UNPLUGGED – A value of 0x00000080.
- DTEVENT_ALL_COMM_EVENTS – A value of 0x0000003F.

Message Constants

The following message constants are defined in the DTCommLib.H file:

- DTMSG_REMOTE_CHAN_OPENED – WM_APP +300.
- DTMSG_REMOTE_CHAN_CLOSED – WM_APP +301.
- DTMSG_ASYNC_DATA_RECEIVED – WM_APP +302.
- DTMSG_SYNC_DATA_RECEIVED – WM_APP +303.
- DTMSG_CHAN_IN_BUFFER_FULL – WM_APP +305.
- DTMSG_OUT_BUFFER_SPACE_AVAILABLE – WM_APP +306.
- DTMSG_ERROR – WM_APP +307.
- DTMSG_MODULE_UNPLUGGED – WM_APP +308.

Counter/Timer Constants

The following register constants are defined in the DTCommonAPI.H file, which is referenced in the DTCommLib.H file:

- COUNTER_RESOLUTION – A value of FFFFFFFFh (2^{32} - 1) when using the internal clock source for counter/timer operations.
- CASCADED_RESOLUTION – A value of FFFFFFFFFFFFFFFFh (2^{64}) when using the cascaded clock source for counter/timer operations.
- COUNTER_CLOCK_FREQ – A value of 18 MHz; the time base for the internal clock for counter/timer operations.
- COUNTER_CLOCK_PERIOD – A value of 55 ns (1/ COUNTER_CLOCK_FREQ).

Flash Memory Constants

The following flash memory constants are defined in the DTCommonAPI.H file:

- USER_DATA_MEM_SIZE – A value of 0x10000
- DATA_FILE_MAGIC_NUMBER – A value of 0x09191956

Enumerated Types

The following enumerated types are defined in the DTCommonAPI.H file, which is referenced in the DTCommLib.H file:

- ADC_RANGE, described on page 123
- AD_DA_CLK_SRC, described on page 124
- CHAN_TYPE, described on page 124
- CLOCK_SRC, described on page 125
- CLOCK_TYPE, described on page 126
- COUNT_MODE, described on page 126
- DAC_FILTER_TYPE, described on page 130
- DAC_RANGE, described on page 131
- EDGE_TYPE, described on page 134
- ERROR_MODE_SELECT, described on page 135
- GATE_TYPE, described on page 135
- POLARITY, described on page 138
- TRIGGER_TYPE, described on page 139

The MODEL_ID enumerated type, described on page 137, is defined in the DTCommLib.H file.

In addition, the DTSTATUS enumerated type, described on page 131, is defined in the DTErrors.H file, which is referenced in the DTCommLib.H file.

ADC_RANGE Enumerated Type

Definition

```
typedef enum
{
  ADC_BIPOLAR_10_VOLTS = 0,
} ADC_RANGE;
```

Include File DTCommonAPI.H

Elements

Name: ADC_BIPOLAR_10_VOLTS

Description: If you specify this element, the analog input range is ±10 V.

AD_DA_CLK_SRC Enumerated Type

Definition

```
typedef enum
{
  AD_DA_CLK_SRC_INTERNAL=0,
  AD_DA_CLK_SRC_EXTERNAL,
  AD_DA_CLK_SRC_SB_MASTER
} AD_DA_CLK_SRC;
```

Include File DTCommonAPI.H

Elements

Name: AD_DA_CLK_SRC_INTERNAL

Description: If you specify this element, the internal sample clock on the DT9840 Series module is used.

You specify the frequency of the internal clock source using the CLOCK_SETUP structure, described on page 145.

Name: AD_DA_CLK_SRC_EXTERNAL

Description: If you specify this element, the signal that is connected to the Ext Clk BNC on the DT9840 Series module is used as the sample clock.

For the DT9841, DT9841E, and DT9841-VIB modules, use an external clock source with a 50% duty cycle. Conversions start on a high-to-low transition of the external clock signal after a rising edge of the internal calibration signal. The resulting frequency of the external clock is always equal to the frequency of the external clock signal that you connected to the module divided by 256 (this division is done internally by the module). For example, if you need a sampling frequency of 100 kHz, use an external clock source with a frequency of 25.6 MHz.

Name: AD_DA_CLK_SRC_SB_MASTER

Description: If you specify this element, the sample clock is derived from the master DT9840 Series module through the Scalable Bus. This is the recommended setting for all slave DT9840 Series modules when you want to synchronize the acquisition of data on all modules.

CHAN_TYPE Enumerated Type

Definition

```
typedef enum
{
  READ_CHAN = 0,
  WRITE_CHAN
} CHAN_TYPE;
```

Include File DTCommonAPI.H

Elements

Name: READ_CHAN

Description: The host computer reads data from the DT9840 Series module.

Name: WRITE_CHAN

Description: The host computer writes data to the DT9840 Series module.

CLOCK_SRC Enumerated Type

Definition

```
typedef enum
{
  INTERNAL_18MHZ = 0,
  EXTERNAL_CLOCK,
  CASCADED_CLOCK
} CLOCK_SRC;
```

Include File DTCommonAPI.H

Elements

Name: INTERNAL_18MHZ

Description: If you specify this element, the internal C/T clock uses a 18 MHz time base. Counter/timer operations start on the rising edge of the clock input signal.

This clock source is not generally used for event counting or up/down counting mode, and is automatically selected if measure mode or continuous measure mode is specified.

Name: EXTERNAL_CLOCK

Description: If you specify this element, an external C/T clock source paces the specified counter/timer channel. Counter/timer operations start on the rising edge of the clock input signal.

This clock source is not generally used for rate generation, one-shot, or repetitive one-shot mode, and is not supported for measure mode or continuous measure mode.

Name: CASCADED_CLOCK

Description: If you specify this element, the clock input signal of the specified counter/timer is connected in software to the clock output signal of the proceeding counter/timer. For example, if you specify CASCADE_CLOCK for counter/timer 2, the clock input of counter/timer 2 is internally connected to the clock output of counter/timer 1.

Note that you cannot specify CASCADED_CLOCK for counter/timer 0, since there is no lower counter/timer with which to cascade.

Description (cont.): The rising edge of the clock input signal is active.

This clock source is not supported for up/down counting, measure mode, or continuous measure mode.

CLOCK_TYPE Enumerated Type

Definition

```
typedef enum
{
  AD_DA_CLOCK,
  AD_CLOCK,
  DA_CLOCK
} CLOCK_TYPE;
```

Include File DTCommonAPI.H

Elements

Name: AD_DA_CLOCK

Description: If you specify this element, the clock that paces both analog input and analog output operations is selected.

Name: AD_CLOCK

Description: If you specify this element, the clock that paces analog input operations is selected.

Name: DA_CLOCK

Description: If you specify this element, the clock that paces analog output operations is selected.

COUNT_MODE Enumerated Type

Definition

```
typedef enum
{
  STD_COUNTING,
  ONE_SHOT,
  REPETITIVE_ONE_SHOT,
  MEASURE,
  UP_DOWN,
  CONTINUOUS_MEASURE
} COUNT_MODE;
```

Include File DTCommonAPI.H

Elements

Name: STD_COUNTING

Description: If you specify this element, the counter/timer performs an event counting or rate generation (pulse output) operation.

For event counting operations, specify an external C/T clock source. For rate generation operations, specify an internal C/T clock source.

Specify either a software gate, normal external gate, or inverted external gate signal to enable the operation.

When the operation is enabled, the counter increments to the value specified by *PulseWidthCount*, described on page 148, and activates the pulse output.

The pulse output stays active until the counter rolls over to 0 (the terminal count). The pulse output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 147.

The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 149. This sequence repeats as long as the counter is enabled by the gate.

You can read the value of the counter at any time using **DT_ReadCTR**.

Name: ONE_SHOT

Description: If you specify this element, the counter/timer outputs a single pulse when the external gate is active.

Specify an internal C/T clock source and either a normal external gate or an inverted external gate signal to enable the operation.

When the operation is enabled, the counter begins incrementing. When the counter increments to the value of *PulseWidthCount*, described on page 148, the value of the counter is output.

The output stays active until the counter rolls over to 0 (the terminal count). When the counter reaches the terminal count, the module clears the one-shot trigger enable bit. The output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 147.

The pulse output then stays inactive, and the counter stays disabled, until the one-shot trigger enable bit is set. The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 149.

Name: REPETITIVE_ONE_SHOT

Description: If you specify this element, the counter/timer outputs a single pulse when the specified external gate signal is active.

Specify an internal C/T clock source and either a normal external gate or an inverted external gate.

When the operation is enabled with an external gate, the counter begins incrementing. When the counter increments to the value of *PulseWidthCount*, described on page 148, the value of the counter is output.

The output stays active until the counter rolls over to 0 (the terminal count). The output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 147.

The user output then stays inactive, and the counter stays disabled, until the next active gate occurs. All gates that occur while the counter is incrementing are ignored. The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 149.

Name: MEASURE

Description: If you specify this element, the counter/timer measures the interval of a signal between a selected measurement start edge and a selected measurement stop edge.

Connect the signal that you want to measure to either the clock input pin or the gate input pin of the counter/timer that you want to use. Then, specify the measurement start edge and the measurement stop edge. Refer to page 134 for more information on the start and stop edges.

When you trigger the operation with **DT_TriggerCTR**, the counter increments from the time it detects the selected start edge until it detects the stop edge. When it detects the selected stop edge, the counter stops counting.

You can determine whether the measure mode operation is complete or not using **DT_ReadCTRStatus**. When the operation is complete, read the value of the counter using **DT_ReadCTR**.

Use the following equations to determine the frequency, period, and pulse width of the signal:

- *Frequency* = 18 MHz/Number of *Counts*
- *Period* = 1/*Frequency*
- Pulse width = Number of Counts/18 MHz

Description (cont.):

Note: In measure mode, the internal C/T clock is used to calculate the interval of the signal between the specified start and stop edges. The pulse width and period count are automatically set to 0. You can specify the polarity of the output signal during and after the measurement.

Name: UP_DOWN

Description: If you specify this element, the counter/timer increments when the specified external gate is enabled and decrements when the specified external gate is disabled. The counter/timer increments or decrements on the rising edge of the external clock input signal.

Use UP_DOWN mode only when the position of the object being monitored stays within the range of the counter, as the operation is not be reliable if the counter increments above FFFFFFFh or decrements below 0.

Specify an external C/T clock source; this mode does not support a cascaded C/T source. Specify either a normal external gate, or inverted external gate signal.

When the operation is enabled, the counter increments to the value specified by *PulseWidthCount*, described on page 148, and activates the pulse output.

The pulse output stays active until the counter rolls over to 0 (the terminal count). The pulse output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 147.

The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 149. This sequence repeats as long as the counter is enabled by the gate.

You can read the value of the counter at any time using **DT_ReadCTR**.

Name: CONTINUOUS_MEASURE

Description: If you specify this element, the counter starts incrementing when it detects a specified start edge and stops incrementing when it detects the next start edge (the stop edge is ignored in this mode). When the operation completes, the counter remains idle until it is next read. On the next read, the current value of the counter (from the previous edge-to-edge measurement operation) is returned and the next edge-to-edge measurement operation is started automatically.

Description (cont.): Connect the signal that you want to measure to either the clock input pin or the gate input pin of the counter/timer that you want to use. Then, specify the measurement start edge. Refer to page 134 for more information on the start edge.

Read the value of the counter using **DT_ReadCTR**. If you read the counter before the measurement is complete, 0 is returned.

Note: In continuous measure mode, the internal C/T clock is used to calculate the interval of the signal between the specified start and stop edges. The pulse width and period count are automatically set to 0.

You can use the following equations to determine the frequency and period of the signal based on the value of the counter:

- *Frequency* = 18 MHz/Number of *Counts*
- *Period* = 1/*Frequency*

DAC_FILTER_TYPE Enumerated Type

Definition

```
typedef enum
{
  NONE = 0,
  FIVE_KHZ,
  TWENTY_KHZ
} DAC_FILTER_TYPE;
```

Include File DTCommonAPI.H

Elements

Name: NONE

Description: If you specify this element, no filter will be applied to the analog output channels. Use this setting for DT9842/2 and DT9842/8 modules.

Name: FIVE_KHZ

Description: If you specify this element, a 5 kHz Bessel reconstruction filter is applied to each analog output channel on the DT9841, DT9841E, and DT9841-VIB. This setting is recommended for slow clock rates.

This element is ignored for the DT9842 and DT9842/8 modules.

Name: TWENTY_KHZ

Description: If you specify this element, a 20 kHz Bessel reconstruction filter is applied to each analog output channel on the DT9841, DT9841E, and DT9841-VIB. This setting is recommended for fast clock rates.

This element is ignored for the DT9842 and DT9842/8 modules.

DAC_RANGE Enumerated Type

Definition

```
typedef enum
{
  DAC_BIPOLAR_10_VOLTS = 0,
  DAC_BIPOLAR_2_PT_5_VOLTS
} DAC_RANGE;
```

Include File DTCommonAPI.H

Elements

Name: DAC_BIPOLAR_10_VOLTS

Description: If you specify this element, the analog output range is ±10 V.

Name: DAC_BIPOLAR_2_PT_5_VOLTS

Description: If you specify this element, the analog output range is ±2.5 V.

DTSTATUS Enumerated Type

Definition

```
typedef enum
{
  DT_SUCCESS = DT_COMMON_BASE_ERR;
  DT_NO_MEMORY;
  DT_BUFFER_TOO_SMALL;
  DT_INVALID_CHAN_NAME;
  DT_INVALID_CHAN_HANDLE;
  DT_INVALID_PHCHAN;
  DT_NOT_IMPLEMENTED;
  DT_INVALID_CALLBACK_PTR;
  DT_INVALID_BUFFER_PTR;
  DT_INVALID_NUM_BYTES;
  DT_INVALID_BYTES_READ_PTR;
  DT_TIMEOUT;
  DT_INVALID_CHAN_TYPE;
  DT_INVALID_POINTER;
  DT_CALLBACK_NOT_FOUND
```

Definition (cont.)

```
DT_CHAN_DISCONNECTED;
DT_INCOMPATIBLE_CHAN_TYPE;
DT_CHAN_ALREADY_OPEN;
DT_CHAN_ALREADY_CLOSED;
 DT_TOO_BIG_FOR_CHAN_BUFFER;
 DT_TOO_BIG_FOR_BOARD_BUFFER;
DT_VOLTAGE_OUT_OF_RANGE;
DT_AD_VALUE_OUT_OF_RANGE;
DT_NO_SYNC_DATA_RECEIVED;
DT_INVALID_DSP_BUFFER_SIZE_PTR;
DT_INVALID_HOST_BUFFER_SIZE_PTR;
DT_INVALID_BUF_SIZE;
DT_CHAN_OPENED_FOR_WRITE;
DT_CHAN_OPENED_FOR_READ;

DT_INVALID_BOARD_NAME =
    DT_HOST_BASE_ERR
DT_INVALID_BOARD_HANDLE;
DT_INVALID_PHBOARD;
DT_INVALID_FILENAME;
DT_INVALID_BOARD_INDEX;
DT_INVALID_BOARD_INFO;
DT_INVALID_WINDOW_HANDLE;
DT_FAILURE;
DT_INVALID_MSG_PTR;
DT_INVALID_MSG_BUF_LEN
DT_INVALID_DSP_ADDRESS;
DT_CANT_OPEN_DRIVER;
DT_CANT_FIND_BOARD;
DT_NO_BOARD_FOR_INDEX;
DT_NO_CHAN_FOR_INDEX;
DT_NO_MESSAGING;
DT_WINDOW_NOT_FOUND;
DT_FILE_NOT_FOUND;
DT_DRIVER_VERSION;
DT_FIRMWARE_VERSION;
DT_DSP_VERSION;
DT_NO_PROGRAM_LOADED;
DT_ERROR_FILE_READ;
DT_ERROR_INVALID_COFF;
DT_INVALID_BUFFER_ADDRESS;
DT_ERROR_MEMORY_MISMATCH;
DT_USB_ERROR;
DT_INVALID_DSP_ADDRESS_PTR;
DT_INVALID_REGISTER;
DT_BOARD_ALREADY_OPENED;
```

Definition (cont.)

```
DT_INVALID_BOARD_OPTIONS;
DT_BOARD_NOT_RESPONDING;
DT_BOARD_WAS_SHUTDOWN;
DT_PROGRAM_TOO_LARGE;
DT_GENERAL_DT9841_ERROR =
     DT_9841_BASE_ERR;
DT_INVALID_SAMPLE_RATE;
DT_NO_DIGITAL_OUT_PORTS;
DT_NO_DIGITAL_IN_PORTS;
DT_IN_LOOP_IS_RUNNING;
DT_OUT_LOOP_IS_RUNNING;
DT_NO_LOOP_RUNNING;
DT_DSP_BIOS_ERROR;
DT_NO_MORE_CHANNELS;
DT_INVALID_VALUE;
DT_INVALID_BLOCK;
DT_INVAID_AD_CHAN;
DT_INVALID_SCAN;
DT_NO_OUTPUTS_ENABLED;
DT_INVALID_COUNTER;
DT_INVALID_SETUP_PTR;
DT_INVALID_MASK;
DT_HANDLER_ALREADY_INSTALLED;
DT_BIT_IS_OUTPUT;
DT_INVALID_COMPARE_VALUE;
DT_COMPARE_IS_OUTPUT;
DT_MASK_IS_OUTPUT;
DT_INVALID_DA_CHAN;
DT_BIT_IS_INPUT;
DT_CLOCK_NOT_RUNNING;
DT_NOT_SB_MASTER;
DT_NOT_SB_SLAVE;
DT_SB_NOT_ENABLED;
DT_INVALID_CLOCK_SRC;
DT_INVALID_COUNT_MODE;
DT_MEASURE_IN_PROGRESS;
DT_ONESHOT_IN_PROGRESS;
DT_SB_NOT_READY;
DT_NO_DAC_FILTER;
DT_SB_IS_BUSY;
DT_SB_TIMEOUT;
DT_INVALID_DMA_BUFFER;
DT_NO_DMA_CHAN;
DT_ERROR_FILTER;
DT_MUST_INIT_LIBRARY;
DT_RANGE_NOT_SUPPORTED;
DT_NO_SB_MODULE_FOR_INDEX;
```

Definition (cont.)

```
  DT_ADC_OVERRUN;
  DT_DAC_UNDERRUN;
  DT_NOT_IEPE_BOARD;
  DT_FLASH_WRITE_FAILED;
  DT_FLASH_VERIFY_FAILED;
  DT_HOST_CHAN_ALREADY_OPEN =
   DT_HOST_COMM_BASE_ERR;
  DT_HOST_CHAN_ALREADY_CLOSED;
  DT_HOST_BUFFER_EMPTY;
  DT_HOST_RCV_BUFFER_FULL;
  DT_HOST_MSG_UNKNOWN_TYPE;
  DT_HOST_MSG_INVALID_LENGTH;
  DT_HOST_CHAN_BUFFER_FULL;
  DT_HOST_SEND_BUFFER_FULL;
  DT_9841_CHAN_ALREADY_OPEN =
   DT_9841_COMM_BASE_ERR;
  DT_9841_CHAN_ALREADY_CLOSED;
  DT_9841_BUFFER_EMPTY;
  DT_9841_RCV_BUFFER_FULL;
  DT_9841_MSG_UNKNOWN_TYPE;
  DT_9841_MSG_INVALID_LENGTH;
  DT_9841_CHAN_BUFFER_FULL;
  DT_9841_SEND_BUFFER_FULL;
} DTSTATUS;
```

Include File DTErrors.H

Elements Each member corresponds to an error code. Refer to Appendix A starting on page 107 and to the *DT9840 Series DSP Library User's Manual* for more information on error codes.

EDGE_TYPE Enumerated Type

Definition

```
typedef enum
{
  SELECTED_GATE_RISES,
  SELECTED_GATE_FALLS,
  EXT_CLOCK_RISES,
  EXT_CLOCK_FALLS
} EDGE_TYPE;
```

Include File DTCommonAPI.H

Elements

Name: SELECTED_GATE_RISES

Description: If you specify this element, a rising edge (low to high transition) on the external gate input pin starts or stops the measurement operation. The software automatically sets the gate type to EXT_NORMAL_GATE in this mode.

Name: SELECTED_GATE_FALLS

Description: If you specify this element, a falling edge (high to low transition) on the external gate input pin starts or stops the measurement operation. The software automatically sets the gate type to EXT_NORMAL_GATE in this mode.

Name: EXTERNAL_CLOCK_RISES

Description: If you specify this element, a rising edge (low to high transition) on the external C/T clock input pin starts or stops the measurement operation.

Name: EXTERNAL_CLOCK_FALLS

Description: If you specify this element, a falling edge (high to low transition) on the external C/T clock input pin starts or stops the measurement operation.

Notes EDGE_TYPE is used only in measure mode and continuous measure mode.

ERROR_MODE_SELECT Enumerated Type

Definition

```
typedef enum {
  STOP_ON_ERROR,
  CONTINUE_ON_ERROR
} ERROR_MODE_SELECT;
```

Include File DTCommonAPI.H

Elements

Name: STOP_ON_ERROR

Description: If you specify this element, the operation stops if the DT9840 Series module detects an error condition.

Name: CONTINUE_ON_ERROR

Description: If you specify this element, the operation continues if the DT9840 Series module detects an error condition.

GATE_TYPE Enumerated Type

Definition

```
typedef enum
{
  GATED_OFF,
  GATED_ON,
  EXT_NORMAL_GATE,
  EXT_INVERTED_GATE
} GATE_TYPE;
```

Include File DTCommonAPI.H

Elements

Name: GATED_OFF

Description: If you specify this element, the gate is disabled.

Name: GATED_ON

Description: If you specify this element, a software command enables the counter/timer operation immediately after execution. This gate type is not supported for up/down counting, measure mode, continuous measure mode, one-shot, or repetitive one-shot operations.

Name: EXT_NORMAL_GATE

Description: For standard counting operations, enables the operation when the external gate signal is high, and disables the operation when the external gate signal is low.

For one-shot and repetitive one-shot operations, enables the operation on the transition from the low level to the high level (rising edge), and disables the operation on the transition from the high level to the low level (falling edge).

For up/down counting mode, increments the counter when the external gate signal is high, and decrements the counter when the external gate signal is low.

For measure mode or continuous measure mode, this value is automatically set to EXT_NORMAL_GATE if you specify the EDGE_TYPE is SELCTED_GATE_RISES or SELECTED_GATE_FALLS.

Name: EXT_INVERTED_GATE

Description: For standard counting operations, enables the operation when the external gate signal is low, and disables the operation when the external gate signal is high.

For one-shot and repetitive one-shot operations, enables the operation on the transition from the high level to the low level (falling edge), and disables the operation on the transition from the low level to the high level (rising edge).

For up/down counting mode, increments the counter when the external gate signal is low, and decrements the counter when the external gate signal is high.

This gate type is not supported for measure mode or continuous measure mode.

MODEL_ID Enumerated Type

Definition

```
typedef enum
{
  DT9841_128_0 = 0,
  DT9841_64_2 = 1,
  DT9842_128_0 = 2,
  DT9842_64_2 = 3,
  DT9848_128_0 = 4,
  DT9848_64_2 = 5,
  DT9841_128_2_6713 = 6,
  DT9842_128_2_6713 = 7,
  DT9848_128_2_6713 = 8,
  DT9841E_128_2_6713 = 9,
  DT9841_128_2_VIB = 10,
  UNKNOWN_MODEL = -1
} MODEL_ID;
```

Include File DTCommLib.H

Elements

Name: DT9841_128_0

Description: This element indicates that you have a DT9841-R module with 128 MBytes of SDRAM and 0 MBytes of flash RAM.

Name: DT9841_64_2

Description: This element indicates that you have a DT9841-F module with 64 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9842_128_0

Description: This element indicates that you have a DT9842/2-R module with 128 MBytes of SDRAM and 0 MBytes of flash RAM.

Name: DT9842_64_2

Description: This element indicates that you have a DT9842/2-F module with 64 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9848_128_0

Description: This element indicates that you have a DT9842/8-R module with 128 MBytes of SDRAM and 0 MBytes of flash RAM.

Name: DT9848_64_2

Description: This element indicates that you have a DT9842/8-F module with 64 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9841_128_2_6713

Description: This element indicates that you have a DT9841 module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9842_128_2_6713

Description: This element indicates that you have a DT9842 module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9848_128_2_6713

Description: This element indicates that you have a DT9848 module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9841E_128_2_6713

Description: This element indicates that you have a DT9841E module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9841_128_2_VIB

Description: This element indicates that you have a DT9841-VIB module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: UNKNOWN_MODEL

Description: This element indicates that you have a newer module type than what is defined in the version of DTCommLib.DLL on your computer. Contact Data Translation for the most up-to-date files.

Notes Non-6713 modules require customized TCF files. If you are using one of these modules and do not have the TCF file you need, contact Technical Support for help.

POLARITY Enumerated Type

Definition

```
typedef enum
{
  ACTIVE_HIGH,
  ACTIVE_LOW
} POLARITY;
```

Include File DTCommonAPI.H

Elements

Name: ACTIVE_HIGH

Description: If you specify this element, the high portion of the total pulse output period is the active portion of the counter/timer clock output signal on the DT9840 Series module.

Name: ACTIVE_LOW

Description: If you specify this element, the low portion of the total pulse output period is the active portion of the counter/timer clock output signal on the DT9840 Series module.

Notes In measure mode, if POLARITY is ACTIVE_HIGH when you call **DT_TriggerCTR**, the counter output is low while the measurement is in progress and high when the measurement is complete. If POLARITY is ACTIVE_LOW when you call **DT_TriggerCTR**, the counter output is high while the measurement is in progress and low when the measurement is complete.

TRIGGER_TYPE Enumerated Type

Definition

```
typedef enum {
  IMMEDIATE=0,
  EXT_TRIGGER=2,
} TRIGGER_TYPE;
```

Include File DTCommonAPI.H

Elements

Name: IMMEDIATE

Description: If you specify this element, the operation on the DT9840 Series module starts immediately when an input or output operation function is called.

Name: EXT_TRIGGER

Description: If you specify this element, a trigger event occurs when the DT9840 Series module detects a low to high transition on the TTL-level signal connected to the Ext Trig BNC on the module. This trigger asserts EXT_INT4 for processing.

Structures

The following structures are defined in the DTCommonAPI.H file, which is referenced in the DTCommLib.H file:

- ADC_SETUP, described on page 140
- BOARD_OPTIONS, described on page 141
- CHAN_OPEN_INFO, described on page 145
- CLOCK_SETUP, described page 145
- CTR_SETUP, described on page 147
- DAC_SETUP, described on page 150
- DATA_FILE_HDR or *PDATA_FILE_HDR, described on page 139
- DIO_SETUP, described on page 152
- IEPE_CHAN_INFO, described on page 153
- IEPE_SETUP, described on page 154
- INPUT_SCAN_BLOCK, described on page 155

- INPUT_SCAN_RCD, described on page 156
- OUTPUT_SCAN_BLOCK, described on page 158
- OUTPUT_SCAN_RCD, described on page 160
- TRIGGER_SETUP, described on page 160

The following structures are defined in the DTCommLib.H file.

- BRD_INFO, described on page 143
- MSG_INFO and PMSG_INFO structures, described on page 158

ADC_SETUP Structure

Definition

```
typedef struct {
  ULONG Termination;
  ERROR_MODE_SELECT ErrorOption;
  ULONG Unused[5];
} ADC_SETUP;
```

Include File DTCommonAPI.H

Elements

Name: Termination

Description: An unsigned integer variable that enables or disables 1 kΩ bias return termination resistors for the analog input channels on the DT9841 and DT9841E modules. Since the DT9841-VIB, DT9842/2 and DT9842/8 modules have single-ended inputs only, this element is ignored for these modules.

This feature is particularly useful with floating signal sources on the DT9841 and DT9841E. Refer to the *DT9840 Series Getting Started Manual* for more information on wiring analog inputs to use bias return termination resistance.

Setting a bit to 1 enables the termination resistor for the associated analog input channel. Setting a bit to 0 disables the termination resistor for the associated analog input channel. The bit values are as follows:

- Bit 1 (0x01) – Corresponds to analog input channel 0.
- Bit 2 (0x02) – Corresponds to analog input channel 1.
- Bit 4 (0x04) – Corresponds to analog input channel 2.
- Bit 8 (0x08) – Corresponds to analog input channel 3.
- Bit 16 (0x10) – Corresponds to analog input channel 4.
- Bit 32 (0x20) – Corresponds to analog input channel 5.
- Bit 64 (0x40) – Corresponds to analog input channel 6.
- Bit 128 (0x80) – Corresponds to analog input channel 7.

Name: ErrorOption

Description: A variable of type ERROR_MODE_SELECT, described on page 135, that specifies whether to continue or stop the operation when an A/D overrun error condition is detected.

If you are performing an input operation and an output operation simultaneously using **DT_ScanLoop**, **DT_BlockLoop**, or **DT_ListLoop**, the following restrictions apply to *ErrorOption*:

- If *ErrorOption* in the DAC_SETUP structure is set to CONTINUE_ON_ERROR, set *ErrorOption* in the ADC_SETUP structure to CONTINUE_ON_ERROR.

  In this case, if either an A/D overrun or D/A underrun error occurs, the user-defined callback functions for both operations are still invoked.

- If *ErrorOption* in the DAC_SETUP structure is set to STOP_ON_ERROR, set *ErrorOption* in the ADC_SETUP structure to STOP_ON_ERROR.

  In this case, if either an A/D overrun or D/A underrun error occurs, both operations stop and the user-defined callbacks are NOT invoked.

Name: Unused

Description: An unsigned integer array of five elements that is currently unused.

Notes In Visual C++, the expression (1 << *ADChan*) generates the appropriate bit value for a given channel.

BOARD_OPTIONS Structure

Definition

```
typedef struct {
  ULONG SizeThisStructure;
  ULONG DSPToHostBufSizeInKb;
  ULONG HostToDSPBufSizeInKb;
  ULONG DbgFlag;
} BOARD_OPTIONS;
```

Include File DTCommonAPI.H

Elements

Name: SizeThisStructure

Description: An unsigned integer variable that specifies the size of the BOARD_OPTIONS structure. This element is provided for future compatibility.

Name: DSPToHostBufSizeInKb

Description: An unsigned integer variable that specifies the size of the module input buffer, in kilobytes, on the host.

If you specify 0 for *DSPToHostBufSizeInKb*, the default size is used (64 kBytes). At a minimum, the size of the module input buffer must be at least as large as the largest message that the DT9840 Series module will send to the host (using the **DT_ChanSyncWrite** or **DT_ChanAsyncWrite** function in the DSP program on the module).

If the amount of memory used by your host application is not a concern, we recommend that you set the size of the module input buffer to a multiple of the largest message (**DT_ChanSyncWrite** or **DT_ChanAsyncWrite**) that the DT9840 Series module will send to the host.

Name: HostToDSPBufSizeInKb

Description: An unsigned integer variable that specifies the size of the module output buffer, in kilobytes, on the host. If you specify 0 for *HostToDSPBufSizeInKb*, the default size (64 kBytes) is used. At a minimum, the size of the module output buffer must be at least as large as the largest message that the host will send to the DT9840 Series module (using the **DT_ChanSyncWrite** or **DT_ChanAsyncWrite** function in the host application).

If the amount of memory used by your host application is not a concern, we recommend that you set the size of the module output buffer to a multiple of the largest message (**DT_ChanSyncWrite** or **DT_ChanAsyncWrite**) that the host will send to the DT9840 Series module.

Name: DbgFlag

Description: An unsigned integer that is passed from the DT9840 Series Host Communication Library to the DT9840 Series DSP Library and is used to control serial debugging on the module.

The DT9840 Series DSP Library uses the lowest 16-bits of this value. The upper 16-bits is available to your host program to define as you wish. For example, you can use this flag on the host to control whether to print messages to the serial port on the module using the **DT_Printf** function of the DT9840 Series DSP Library.

You can read the value of this flag in the DSP program on the module using the **DT_GetDebugFlag** function in the DT9840 Series DSP Library.

Notes Ensure that the combined size of *DSPToHostBufSizeInKb* and *HostToDSPBufSizeInKb* does not exceed 48 MBytes. In addition, ensure that you have enough host memory and free RAM on the DT9840 Series module to accommodate the sizes specified in *DSPToHostBufSizeInKb* and *HostToDSPBufSizeInKb.*

Notes (cont.) If you specify NULL for the *pBoardOptions* parameter of the **DT_BoardOpen** function, the module input and output buffers are set to their default sizes (64 kBytes).

BRD_INFO Structure

Definition

```
typedef struct {
  MODEL_ID Model;
  CHAR *pModelNameStr;
  ULONG MB_RAM;
  ULONG MB_Flash;
  BOOL ProgramIsRunning;
  ULONG FPGAVersion;
  ULONG USBPortType;
  BOOL SBEnabled;
  LONG SBAddress;
  BOOL SBTerminated;
  ULONG Reserved[15];
} BRD_INFO;
```

Include File DTCommLib.H

Elements

Name: Model

Description: A variable of type MODEL_ID, described on page 137, that represents the module type.

Name: pModelNameStr

Description: A pointer to a variable of type CHAR in which the name of the module is returned. You specify this name when you configure the device driver. Refer to the *DT9840 Series Getting Started Manual* for more information on configuring the device driver.

Name: MB_RAM

Description: An unsigned integer variable that represents the size of the SDRAM on the module, in megabytes.

Name: MB_Flash

Description: An unsigned integer variable that represents the size of the flash on the module, in megabytes.

Name: ProgramIsRunning

Description: A Boolean variable that indicates whether a DSP program is currently running on the DT9840 Series module.

If TRUE, a DSP program is currently running on the specified DT9840 Series module. If FALSE, a DSP program is not currently running on the specified DT9840 Series module.

Name: FPGAVersion

Description: An unsigned integer variable in which the version of the FPGA code on the module is returned.

Name: USBPortType

Description: An unsigned integer variable in which the version of the USB port that is used is returned.

Name: SBEnabled

Description: A Boolean variable that indicates whether the Scalable Bus is enabled for the module. If TRUE, the Scalable Bus is enabled for the module. If FALSE, the Scalable Bus is not enabled for the module.

Refer to the *DT9840 Series Getting Started Manual* for more information on enabling the Scalable Bus using the DT9840 Series Control Panel applet.

Name: SBAddress

Description: An integer variable that contains the Scalable Bus address of the module. If the module is the master, a bus address of 0 is returned. If the module is a slave, a bus address of 1 through 3 is returned.

Refer to the *DT9840 Series Getting Started Manual* for more information on assigning addresses to modules using the DT9840 Series Control Panel applet.

Name: SBTerminated

Description: A Boolean variable that indicates whether the Scalable Bus is terminated for the module. If TRUE, the Scalable Bus is terminated for the module. If FALSE, the Scalable Bus is not terminated for the module.

Refer to the *DT9840 Series Getting Started Manual* for more information on terminating the Scalable Bus using the DT9840 Series Control Panel applet.

Name: Reserved

Description: An unsigned integer array of 15 elements that is reserved for future use.

CHAN_OPEN_INFO Structure

Definition

```
typedef struct {
  CHAN_NAME ChanName;
  CHAN_TYPE RemoteChanType;
  ULONG RemoteBufSize;
} CHAN_OPEN_INFO;
```

Include File DTCommonAPI.H

Elements

Name: ChanName

Description: A variable of type CHAN_NAME, described on page 116, that represents the name of the communication channel on the host.

Name: RemoteChanType

Description: A variable of type CHAN_TYPE, described on page 124, that determines the whether the communication channel on the host was opened for reading or for writing.

Name: RemoteBufSize

Description: An unsigned integer variable that represents the size of the channel input buffer on the host.

CLOCK_SETUP Structure

Definition

```
typedef struct {
  AD_DA_CLK_SRC ClockSource;
  DOUBLE SampleRate;
  DOUBLE MaxSampleRate;
  DOUBLE MinSampleRate;
  DOUBLE ActualSampleRate;
  CLOCK_TYPE ClockType;
  ULONG Unused[5];
} CLOCK_SETUP;
```

Include File DTCommonAPI.H

Elements

Name: ClockSource

Description: A variable of type AD_DA_CLK_SRC, described on page 124, that specifies the clock source for the input/output operation.

Name: SampleRate

Description: A variable of type DOUBLE that specifies the frequency, in Hz, for the internal sample clock. For the DT9841, DT9841E, and DT9841-VIB modules, values range from 200 Hz to 100,000 Hz. For the DT9842/2 and DT9842/8 modules, values range from 0 Hz to 100,000 Hz.

Description (cont.): The actual frequency that the module can achieve may be slightly different than the frequency you specified, due to the accuracy of the clock frequency (0.01%). The actual clock frequency is returned in *ActualSampleRate*.

For the DT9841, DT9841E, and DT9841-VIB modules, the value that you specify for the internal clock frequency is multiplied by 512 internally to set the oscillator on the module. The resulting signal from the oscillator is then divided by 2 to provide a clock signal to the A/D and D/A converters that is oversampled 256 times and has a 50% duty cycle. For example, if you specify an internal clock frequency of 100 kHz, internally the DT9841, DT9841E, and DT9841-VIB modules set the oscillator to 51.2 MHz then divides the resulting signal by 2 to provide a 25.6 MHz signal with a 50% duty cycle to the A/D and D/A converters.

Name: MaxSampleRate

Description: A variable of type DOUBLE that indicates the maximum frequency, in Hz, of the internal sample clock on the DT9840 Series module. This value is returned by **DT_SetupClock**, described in the *DT9840 Series DSP Library User's Manual*. For all DT9840 Series modules, this value is always 100,000 Hz.

Name: MinSampleRate

Description: A variable of type DOUBLE that indicates the minimum frequency, in Hz, of the internal sample clock on the DT9840 Series module. This value is returned by **DT_SetupClock**, described in the *DT9840 Series DSP Library User's Manual*. For the DT9841, DT9841E, and DT9841-VIB, this value is 200 Hz. For the DT9842/2 and DT9842/8, this value is 0 Hz.

Name: ActualSampleRate

Description: A variable of type DOUBLE that indicates the actual sample frequency, in Hz, that the DT9840 Series module could achieve. This value is returned by **DT_SetupClock**, described in the *DT9840 Series DSP Library User's Manual*.

Name: ClockType

Description: A variable of type CLOCK_TYPE, described on page 126, that indicates the clock to use.

Name: Unused

Description: An unsigned integer array of five elements that is currently unused.

CTR_SETUP Structure

Definition

```
typedef struct {
  COUNT_MODE CountMode;
  CLOCK_SRC ClkSrc;
  ULONG PeriodCount;
  ULONG PulseWidthCount;
  POLARITY OutputPolarity;
  GATE_TYPE GateType;
  EDGE_TYPE MeasureModeStartEdge;
  EDGE_TYPE MeasureModeStopEdge;
  ULONG Unused[5];
} CTR_SETUP;
```

Include File DTCommonAPI.H

Elements

Name: CountMode

Description: A variable of type COUNT_MODE, described on page 126, that defines the operation of the specified counter/timer.

Name: ClkSrc

Description: A variable of type CLOCK_SRC, described on page 125, that defines the clock input source for the counter/timer.

In measure mode and continuous measure mode, the software automatically sets this value to INTERNAL_18MHz.

Name: PeriodCount

Description: An unsigned integer variable that defines the initial value that is loaded into the specified counter/timer. Values range from 1 to FFFFFFFFh. The counter/timer counts from *PeriodCount* to FFFFFFFFh, rolls over to 0 (the terminal count), and then reloads *PeriodCount*.

In event counting operations, this variable determines the value at which to start counting.

In pulse output operations, this variable determines the frequency of the pulse output signal.

This variable is automatically set to 0 if measure mode or continuous measure mode is specified.

Use the following equations to determine the value for *PeriodCount* when performing pulse output operations; refer to page 122 for more information on the constants used in these equations:

- *PeriodCount* = COUNTER_RESOLUTION + 2 – (COUNTER_CLOCK_FREQ / output frequency)

  For example, if the clock input frequency is 18 MHz and the desired pulse output frequency is 6.0 MHz, the required *PeriodCount* is FFFFFFFEh.

Description (cont.):

- #clks per output period = COUNTER_RESOLUTION + 2 – *PeriodCount*

  For example, if *PeriodCount* = FFFFFFFEh, three clocks occur for each output period. In this example, the counter counts FFFFFFFEh, FFFFFFFFh, 0, FFFFFFFEh, FFFFFFFFh, 0, and so on.

- output period = (# clks per output period) x COUNTER_CLOCK_PERIOD

  For example, if three clocks occur for each output period and the clock period is 55.555 ns, then the output period is 166.666 ns.

- output frequency = 1 / output period

  For example, if the output period is 166.666 ns, then the output frequency is 6.0 MHz.

  The maximum output frequency is 9 MHz. The minimum output frequency is 0.0042 Hz.

Name: PulseWidthCount

Description: An unsigned integer variable that determines when the output pulse of the counter/timer is activated; this variable determines the duty cycle of the pulse output signal. Values range from 1 to FFFFFFFFh.

In addition to rate generation, one-shot, and repetitive one-shot operations, you can use this value to determine when to stop counting in event counting applications.

This variable is automatically set to 0 if measure mode or continuous measure mode is specified.

Use the following equations to determine the value for *PulseWidthCount*:

- *PulseWidthCount* = COUNTER_RESOLUTION + 1 – (duty cycle x (clock frequency /output frequency))

  For example, if the clock input frequency is 18 MHz, the frequency of the output pulse is 6.0 MHz, and the desired duty cycle is .66 (66%), then the required *PulseWidthCount* is FFFFFFFEh.

- #clks per output pulse = COUNTER_RESOLUTION + 1 – *PulseWidthCount*

  For example, if *PulseWidthCount* = FFFFFFFEh, two clocks occur for each output pulse. In this example, the output is active for counts FFFFFFFFh and 0.

Description (cont.):

- output pulse width = (#clks per output pulse) x COUNTER_CLOCK_PERIOD

  For example, if the clock period is 55.55 ns and two clock occur for each output pulse, then the output pulse width is 111.11 ns.

- output duty cycle = (output pulse width) / (output period)

  For example, if the output pulse width is 222.2 ns and the output pulse period is 333.3 ns, then the output duty cycle is .66 (66%).

Name: OutputPolarity

Description: A variable of type POLARITY, described on page 138, that defines the polarity of the output pulse for the specified counter/timer.

In measure mode, if POLARITY is ACTIVE_HIGH when you call **DT_TriggerCTR**, the counter output is low while the measurement is in progress and high when the measurement is complete. If POLARITY is ACTIVE_LOW when you call **DT_TriggerCTR**, the counter output is high while the measurement is in progress and low when the measurement is complete.

Name: GateType

Description: A variable of type GATE_TYPE, described on page 135, that defines the type of gate to use for the specified counter/timer.

For measure or continuous measure mode, this value is automatically set to EXT_NORMAL_GATE if you specify the EDGE_TYPE is SELCTED_GATE_RISES or SELECTED_GATE_FALLS.

Name: MeasureModeStartEdge

Description: Used for measure mode and continuous measure mode operations only, a variable of type EDGE_TYPE, described on page 134, that specifies what edge of the signal connected to the external gate or external clock input pin starts the measurement.

In continuous measure mode, the counter increments from the first start edge to the next start edge; the stop edge is ignored.

This variable is ignored for all other counter/timer operations.

Name: MeasureModeStopEdge

Description: Used for measure operations only, a variable of type EDGE_TYPE, described on page 134, that specifies what edge of the signal connected to the external gate or external clock input pin stops the measurement. This variable is ignored for all other counter/timer operations.

Name: Unused[5]

Description: An unsigned integer array with five elements that is currently unused.

DAC_SETUP Structure

Definition

```
typedef struct {
  DAC_FILTER_TYPE FilterType;
  LONG DAInitialValue[
   NUM_DAC_CHANS];
  ERROR_MODE_SELECT ErrorOption;
  DAC_RANGE DacRange;
  ULONG Unused[5];
} DAC_SETUP;
```

Include File DTCommonAPI.H

Elements

Name: FilterType

Description: A variable of type DAC_FILTER_TYPE, described on page 130, that specifies the reconstruction filter to apply to the analog output channels on the DT9840 Series module.

Name: DAInitialValue

Description: For clocked analog output operations, an array of integer variables that specifies the initial raw counts to output to the analog output channels when **DT_SetupDAC**, described in the *DT9840 Series DSP Library User's Manual*, is called.

Name: ErrorOption

Description: A variable of type ERROR_MODE_SELECT, described on page 135, that specifies whether to continue or stop the analog output operation when an D/A underrun error is detected.

If you are performing an output operation and an input operation simultaneously using **DT_ScanLoop**, **DT_BlockLoop**, or **DT_ListLoop**, the following restrictions apply to *ErrorOption*:

- If *ErrorOption* in the ADC_SETUP structure is set to CONTINUE_ON_ERROR, set *ErrorOption* in the DAC_SETUP structure to CONTINUE_ON_ERROR.

  In this case, if either a D/A underrun or A/D overrun error occurs, the user-defined callback functions for both operations are still invoked.

Description (cont.):

- If *ErrorOption* in the ADC_SETUP structure is set to STOP_ON_ERROR, set *ErrorOption* in the DAC_SETUP structure to STOP_ON_ERROR.

  In this case, if either a D/A underrun or A/D overrun error occurs, both operations stop and the user-defined callbacks are NOT invoked.

Name: DacRange

Description: A variable of type DAC_RANGE, described on page 131, that specifies the output range for the analog output channels.

Name: Unused[5]

Description: An unsigned integer array of five elements that is currently unused.

Notes On the DT9842/2, the two analog output channels are 180 degrees out of phase with each other. On the DT9842/8, each analog output channel is 45 degrees out of phase with the previous channel.

To convert a voltage to a raw initial value, use **DT_VoltsToDAValue**, described on page 81.

We recommend that you turn on the 20 kHz filter whenever you want the smoothest possible analog output signal; this is particularly true at clock frequencies lower than 50 kHz.

DATA_FILE_HDR or *PDATA_FILE_HDR Structure

Definition

```
typedef struct {
  ULONG DataFileID;
  ULONG NumBytes;
} DATA_FILE_HDR, *PDATA_FILE_HDR;
```

Include File DTCommonAPI.H

Elements

Name: DataFileID

Description: An unsigned integer variable that identifies the data file. This variable must be set to the value of DATA_FILE_MAGIC_NUMBER, defined on page 123.

Name: NumBytes

Description: An unsigned integer variable that specifies the number of bytes to write to or read from the 64 KB reserved block of flash memory. This value should be greater than 0 and less than USER_DATA_MEM_SIZE, defined on page 123.

Notes This structure is used to define the header of each data file to write to or read from flash memory using the DT9840 Series Flash Programmer Utility. The header is followed by the actual data to write to or read from the 64 KB reserved block in flash memory.

DIO_SETUP Structure

Definition

```
typedef struct {
  BOOL ClockDout;
  BOOL Port0Out;
  BOOL Port1Out;
  BOOL Port2Out;
  BOOL DeglitchPort0;
  ULONG InitialDOValue;
  ULONG Unused[5];
} DIO_SETUP;
```

Include File DTCommonAPI.H

Elements

Name: ClockDout

Description: A Boolean value that specifies whether to clock digital output values from the DT9840 Series module while analog output values are being clocked out.

If TRUE, digital output values are output with analog output values at the frequency of the sample clock when any of the following functions in the DT9840 Series DSP Library is called:

- DT_ScanLoop
- DT_BlockLoop
- DT_FunctionGen
- DT_ListLoop
- DT_WriteDACDIO

If FALSE, digital output values are not clocked out when analog output values are clocked out.

Name: Port0Out

Description: A Boolean variable that specifies whether digital port 0 on the DT9840 Series module is an input or an output port. If FALSE, digital port 0 is an input port; if TRUE, digital port 0 is an output port.

Name: Port1Out

Description: A Boolean variable that specifies whether digital port 1 on the DT9840 Series module is an input or an output port. If FALSE, digital port 1 is an input port; if TRUE, digital port 1 is an output port.

Name: Port2Out

Description: A Boolean variable that specifies whether digital port 2 on the DT9840 Series module is an input or an output port. If FALSE, digital port 2 is an input port; if TRUE, digital port 2 is an output port.

Name: DeglitchPort0

Description: A Boolean variable that specifies whether to enable or disable deglitch circuitry on digital port 0 of the DT9840 Series module; this function minimizes false triggers due to noise and is particularly useful when using the interrupt-on-change feature.

If TRUE, a 10 ms deglitch function is enabled for digital port 0. If FALSE, the 10 ms deglitch function is disabled for digital port 0.

Note that if you are using the interrupt-on-change feature when deglitching is enabled, the maximum frequency that any change can be detected is 500 Hz; if deglitching is disabled, the maximum frequency that any change can be detected is is 100 kHz.

This parameter is ignored if *Port0Out* is TRUE.

Name: InitialDOValue

Description: An unsigned integer variable that specifies the initial value to output to the digital output lines when **DT_SetupDIO**, described in the *DT9840 Series DSP Library User's Manual*, is called.

Name: Unused[5]

Description: An unsigned integer array of five elements that is currently unused.

IEPE_CHAN_INFO Structure

Definition

```
typedef struct {
  BOOL CurrentSourceEnabled;
  BOOL FilterEnabled;
  BOOL ACCoupling;
} IEPE_CHAN_INFO;
```

Include File DTCommonAPI.H

Elements

Name: CurrentSourceEnabled

Description: A Boolean variable that indicates whether the 4 mA current source is enabled or disabled for the analog input channel.

If the value is TRUE, the 4 mA current source is enabled. If the value is FALSE, the 4 mA current source is disabled.

Name: FilterEnabled

Description: A Boolean variable that indicates whether the 10 kHz filter is enabled or disabled for the analog input channel.

If the value is TRUE, the 10 kHz filter is enabled. If the value is FALSE, the 10 kHz filter is disabled.

Name: ACCoupling

Description: A Boolean variable that indicates whether AC coupling is enabled or disabled for the analog input channel.

If the value is TRUE, AC coupling is enabled. If the value is FALSE, AC coupling is disabled, and DC coupling is used.

Notes This structure is used by the IEPE_SETUP structure, described on page 154, and applies to the DT9841-VIB module only.

IEPE_SETUP Structure

Definition

```
typedef struct {
  IEPE_CHAN_INFO Chan[NUM_ADC_REGS];
} IEPE_SETUP;
```

Include File DTCommonAPI.H

Elements

Name: Chan[NUM_ADC_REGS]

Description: An array of type IEPE_CHAN_INFO, described on page 153, that specifies the IEPE parameters for all of the analog input channels on the DT9841-VIB module.

The array is defined to be as large as the number of analog input registers. Refer to page 117 or more information on NUM_ADC_REGS.

Notes This structure applies to the DT9841-VIB module only.

INPUT_SCAN_BLOCK Structure

Definition

```
typedef struct {
  struct INPUT_SCAN_BLOCK *pNext;
  ULONG nScansThisBlock;
  ULONG AllocSize;
  ULONG BlockNum;
  BOOL BlockComplete;
  INPUT_SCAN_RCD Scans[1];
} INPUT_SCAN_BLOCK;
```

Include File DTCommonAPI.H

Elements

Name: pNext

Description: A pointer of type INPUT_SCAN_BLOCK that points to the next block in the list.

For block loop operations, the value of *pNext* is NULL. For list loop operations, the value of *pNext* is passed to this structure when **DT_MallocInputBlockList**, described in the *DT9840 Series DSP Library User's Manual*, is called.

If the list was allocated for a continuous list loop operation, the last block's *pNext* pointer points to the first block in the list. If the list was allocated for a one-pass list loop operation, the last block's *pNext* pointer is NULL, and the input list operation stops when the last block has been acquired.

Name: nScansThisBlock

Description: An unsigned integer variable that specifies the number of scans to perform for each block. For example, if *nScansThisBlock* is 1, the block of data that is acquired represents one scan of all the analog input channels and digital input lines. If *nScansThisBlock* is 100, the block of data that is acquired represents 100 scans of all the analog input channels and digital input lines.

For block loop operations, the value of *nScansThisBlock* is passed to this structure when **DT_BlockLoop**, described in the *DT9840 Series DSP Library User's Manual*, is called.

For list loop operations, the value of *nScansThisBlock* is passed to this structure when **DT_MallocInputBlockList**, described in the *DT9840 Series DSP Library User's Manual*, is called.

Name: AllocSize

Description: An unsigned integer variable in which the size of the input scan block, in bytes, is returned.

Note: The value of this parameter should never be modified programmatically.

Name: BlockNum

Description: An unsigned integer variable that specifies the number of blocks in the INPUT_SCAN_BLOCK structure.

For block loop operations, the value of this variable is always 1.

For list loop operations, the value of *BlockNum* is passed to this structure when **DT_MallocInputBlockList**, described in the *DT9840 Series DSP Library User's Manual*, is called.

Name: BlockComplete

Description: A Boolean variable that indicates when the current block is filled. If the value is TRUE, the current block is filled. If the value is FALSE, the current block is not filled.

Once the block is processed, you must set this variable to FALSE.

Name: Scans[1]

Description: An array of type INPUT_SCAN_RCD, described on page 156, in which the analog input, digital input, and counter/timer data is returned. Each scan is an element in the array. The number of elements in the array is specified by *nScansThisBlock.*

INPUT_SCAN_RCD Structure

Definition

```
typedef struct {
  LONG ADCValues[NUM_ADC_REGS];
  ULONG FlagsAndDin;
  ULONG CounterValues[
   NUM_COUNTERS];
} INPUT_SCAN_RCD;
```

Include File DTCommonAPI.H

Elements

Name: ADCValues[NUM_ADC_REGS]

Description: An integer array in which the acquired value of all the analog input channels is returned.

The array is defined to be as large as the number of analog input registers. Refer to page 117 for more information on NUM_ADC_REGS.

Name: FlagsandDin

Description: An unsigned integer value in which the state of the digital input ports and the eight hardware flags is returned.

Bits 0 to 7 represent port 0, bits 8 to 15 represent port 1, bits 9 to 23 represent port 2, and bits 24 to 31 represent the following hardware flags:

- Bit 24

  1 = Temperature is 65° C or over.
  0 = Temperature is under 65° C.

- Bit 25

  1 = Counter/timer 2 has overflowed.
  0 = Counter/timer 2 has not overflowed.

- Bit 26

  1 = Counter/timer 1 has overflowed.
  0 = Counter/timer 1 has not overflowed.

- Bit 27

  1 = Counter/timer 0 has overflowed.
  0 = Counter/timer 0 has not overflowed.

- Bit 28

  1 = A FIFO Full error occurred on the Scalable Bus.
  0 = A FIFO Full error did not occur on the Scalable Bus.

- Bit 29

- 1 = An analog output trigger occurred.
  0 = An analog output trigger did not occur.

- Bit 30

  1 = An analog input trigger error occurred.
  0 = An analog input trigger error did not occur.

- Bit 31

  1 = An analog input trigger occurred.
  0 = An analog input trigger did not occur.

Name: CounterValues[NUM_COUNTERS]

Description: An unsigned integer array in which the acquired value of all the counter/timer channels is returned.

The array is defined to be as large as the number of counter/timer channels. The DT9840 Series modules support three counter/timer channels (0 to 2).

MSG_INFO and PMSG_INFO Structures

Definition

```
typedef struct {
  PVOID pMessage;
  ULONG NumBytes;
} MSG_INFO, *PMSG_INFO;
```

Include File DTCommLib.H

Elements

Name: pMessage

Description: A pointer to a variable of type VOID that either specifies the message to send or receives the message that is sent.

Name: NumBytes

Description: An unsigned integer variable that identifies the number of bytes in the message to send or that have been received.

OUTPUT_SCAN_BLOCK Structure

Definition

```
typedef struct {
  struct OUTPUT_SCAN_BLOCK *pNext;
  ULONG nScansThisBlock;
  ULONG AllocSize;
  ULONG BlockNum;
  BOOL BlockComplete;
  OUTPUT_SCAN_RCD Scans[1];
} OUTPUT_SCAN_BLOCK;
```

Include File DTCommonAPI.H

Elements

Name: pNext

Description: Specifies a pointer of type OUTPUT_SCAN_BLOCK that points to the next block in the list.

For block loop operations, the value of *pNext* is NULL.

Description (cont.): For list loop operations, the value of *pNext* is passed to this structure when **DT_MallocOutputBlockList**, described in the *DT9840 Series DSP Library User's Manual*, is called. If the list was allocated for a continuous list loop operation, the last block's *pNext* pointer points to the first block in the list. If the list was allocated for a one-pass list loop operation, the last block's *pNext* pointer is NULL, and the list output operation stops when the last block has been output.

Name: nScansThisBlock

Description: An unsigned integer variable that specifies the number of scans in each block. For example, if *nScansThisBlock* is 1, the block of data that is output represents one scan of all the analog output channels and digital output lines. If *nScansPerBlock* is 100, the block of data that is output represents 100 scans of all the analog output channels and digital output lines.

For block loop operations, the value of *nScansThisBlock* is passed to this structure when **DT_BlockLoop**, described in the *DT9840 Series DSP Library User's Manual*, is called.

For list loop operations, the value of *nScansThisBlock* is passed to this structure when **DT_MallocOutputBlockList**, described in the *DT9840 Series DSP Library User's Manual*, is called.

Name: AllocSize

Description: An unsigned integer variable in which the size of the output scan block, in bytes, is returned.

Note: The value of this parameter should never be modified programmatically.

Name: BlockNum

Description: A unsigned integer variable that specifies the number of blocks in the OUTPUT_SCAN_BLOCK structure.

For block loop operations, the value of this variable is always 1.

For list loop operations, the value of *BlockNum* is passed to this structure when **DT_MallocOutputBlockList**, described in the *DT9840 Series DSP Library User's Manual*, is called.

Name: BlockComplete

Description: A Boolean variable that indicates when the current block is empty. If the value is TRUE, the current block is empty. If the value is FALSE, the current block is not empty.

Once the block has been processed, you must set this variable to FALSE.

Name: Scans[1]

Description: An array of type OUTPUT_SCAN_RCD, described on page 160, that contains the analog output and digital data to output for each scan. Each scan is an element in the array. The number of elements in the array is specified by *nScansThisBlock*.

OUTPUT_SCAN_RCD Structure

Definition

```
typedef struct {
  LONG DACValues[NUM_DAC_CHANS];
  ULONG DoutValues;
} OUTPUT_SCAN_RCD;
```

Include File DTCommonAPI.H

Elements

Name: DACValues[NUM_DAC_CHANS]

Description: An integer array of analog values to output to the analog output channels.

The array is defined to be as large as the number of analog output channels on the module.

Name: DoutValues

Description: An unsigned integer variable that contains a 24-bit value to output to the digital output lines. Bits 0 to 7 represent port 0, bits 8 to 15 represent port 1, and bits 9 to 23 represent port 2. (Note that for this value to be output, you first need to call **DT_SetupDIO** and set *ClockDout* to TRUE).

TRIGGER_SETUP Structure

Definition

```
typedef struct {
  TRIGGER_TYPE TrigType;
  ULONG Unused1;
  ULONG Unused2;
} TRIGGER_SETUP;
```

Include File DTCommonAPI.H

Elements

Name: TrigType

Description: A variable of type TRIGGER_TYPE, described on page 139, that specifies the trigger mode on the DT9840 Series module.

Name: Unused1

Description: An unsigned integer variable that is currently unused; specify 0 for this element.

Name:	Unused2
Description:	An unsigned integer variable that is currently unused; specify 0 for this element.

Callback Functions

The CALLBACKPROC callback function is defined in the DTCommonAPI.H file as follows:

CALLBACKPROC CallBack Function

Definition

```
BOOL *CALLBACKPROC(
   ULONG EventCode,
   VOID *pContext,
   LONG Param1,
   LONG Param2
   );
```

Include File DTCommonAPI.H

Parameters

Name: EventCode

Description: An unsigned integer variable, returned by this function, that represents the type of event that occurred.

The following constants are defined for each event type:

- DTEVENT_REMOTE_CHAN_OPENED, described on page 215.
- DTEVENT_REMOTE_CHAN_CLOSED, described on page 214.
- DTEVENT_ASYNC_DATA_RECEIVED, described on page 208.
- DTEVENT_SYNC_DATA_RECEIVED, described on page 215.
- DTEVENT_CHAN_IN_BUFFER_FULL, described on page 209.
- DTEVENT_OUT_BUFFER_SPACE_AVAILABLE, described on page 213.
- DTEVENT_ERROR, described on page 210.
- DTEVENT_MODULE_UNPLUGGED, described on page 212. This event occurs only on the host.
- DTEVENT_INPUT_SCAN_BLOCK_OVERRUN, described on page 211.
- DTEVENT_OUTPUT_SCAN_BLOCK_UNDERRUN, described on page 214.
- DTEVENT_AD_OVERRUN, described on page 208.
- DTEVENT_DA_UNDERRUN, described on page 210.

Name: pContext

Description: A pointer to a variable of type VOID that is passed to this function when **DT_RegisterCallback**, described on page 70, is called.

A typical use for *pContext* is to pass a "this" pointer in C++ so that the callback function has access to the object that registered the callback.

If the events DTEVENT_INPUT_SCAN_BLOCK_OVERRUN, DTEVENT_OUTPUT_SCAN_BLOCK_UNDERRUN, DTEVENT_AD_OVERRUN, or DTEVENT_DA_UNDERRUN occur, pContext must point to a global variable, constant, or dynamically allocated block, and NOT to a variable on a stack. These events can only occur in a DSP program.

Name: Param1

Description: An integer variable, returned by this function, whose value depends on the event that occurred. Refer to Appendix C starting on page 207 for more information.

Name: Param2

Description: An integer variable, returned by this function, whose value depends on the event that occurred. Refer to Appendix C starting on page 207 for more information.

For Visual Basic Programmers

This section describes the global constants, enumerated types, and data types that are used when calling the DT9840 Series host communication functions in Visual Basic. Callback functions for the DT9840 Series modules are not supported in Visual Basic.

Constants

The following constants are provided for the DT9840 Series Host Communication Library in Visual Basic:

- Global constants, described on this page
- Conversion constants, described on page 165
- Register constants, described on page 166
- Error constants, described on page 169
- Counter/timer constants, described on page 169
- Flash memory constants, described on page 169

Global Constants

The following global constants are defined in the DTCommLib.BAS file:

- DTCommLibBasVer – The current version of the DTCommLib.BAS file.
- DT_BOARD_TYPE – The specific DT9840 Series module; this value is 9841 to identify the DT9841 and DT9841-VIB, 98412 to identify the DT9841E module, 9842 to identify the DT9842/2, or 9848 to identify the DT9842/8.
- NUM_ADC_REGS – The number of analog input registers. All DT9840 Series modules have 8 analog input registers. For the DT9841E, the first two analog input registers correspond to analog input channels 0 and 1; the data in the remaining analog input registers should be ignored.
- NUM_ADC_CHANS – The number of analog input channels. For the DT9841, DT9841-VIB, DT9842/2, and DT9842/8, NUM_ADC_CHAN is 8. For the DT9841E, NUM_ADC_CHAN is 2.
- NUM_COUNTERS – The number of counter/timer channels; for all DT9840 Series modules, this value is 3.
- NUM_DIN – The number of 24-bit digital input lines; for all DT9840 Series modules, this value is 1. For the DT9841-VIB, only the first 16-bits are accessible; the 8-remaining bits should be ignored.
- NUM_DOUT – The number of 24-bit digital output lines; for all DT9840 Series modules, this value is 1. For the DT9841-VIB, only the first 16-bits are accessible; the 8-remaining bits should be ignored.
- NUM_DAC_CHANS – The number of analog output channels; for the DT9841, DT9841E, DT9841-VIB, and DT9842/2, this value is 2; for the DT9842/8, this value is 8.

- NUM_STEPS – The number of steps for the A/D converter on the module. For the DT9841, DT9841E, and DT9841-VIB, this value is 2^{24}; for the DT9842/2 and DT9842/8, this value is 2^{16}.
- COUNTS_PER_STEPS – The number of count changes for each step of the A/D converter. For the DT9841, DT9841E, and DT9841-VIB, this value is 2^{8}; for the DT9842/2 and DT9842/8, this value is 2^{16}.
- CLOCK_SETUP_LEN – The size of the CLOCK_SETUP data type in bytes (8 *4 + 4 *8).
- TRIGGER_SETUP_LEN – The size of the TRIGGER_SETUP data type in bytes (3 *4).
- ADC_SETUP_LEN – The size of the ADC_SETUP data type in bytes (7 *4).
- DAC_SETUP_LEN – The size of the DAC_SETUP data type in bytes ((8+NUM_DAC_CHANS) * 4).
- DIO_SETUP_LEN – The size of the DIO_SETUP data type in bytes (5*4 + 12*2).
- CTR_SETUP_LEN – The size of the CTR_SETUP data type in bytes (13 *4).
- INPUT_SCAN_RCD_LEN – The size of the INPUT_SCAN_RCD data type in bytes (12 *4).
- OUTPUT_SCAN_RCD_LEN – The size of the OUTPUT_SCAN_RCD data type in bytes ((NUM_DAC_CHANS+1) * 4.
- NUM_INPUTS_IN_RCD – The number of analog input registers (NUM_ADC_REGS), number of counter/timer channels (NUM_COUNTERS), and number of 24-bit digital input lines (NUM_DIN) in the input scan record for the specified module type. For example, for the DT9841, NUM_INPUTS_IN_RCD is 12 (8 analog input registers, 3 counter/timer channels, and 1 24-bit digital input port).
- NUM_OUTPUTS_IN_RCD – The number of analog output channels (NUM_DAC_CHANS) and number of 24-bit digital output lines (NUM_DOUT) in the output scan record for the specified module type. For example, for the DT9841, NUM_INPUTS_IN_RCD is 3 (2 analog output channels and 1 24-bit digital output port).

Conversion Constants

The following conversion constants are defined in the DTCommLib.BAS file:

- BIP10_VOLTS_SPAN – An input or output range of ±10 V; this is 20# (LONG).
- BIP2PT5_VOLTS_SPAN – An input or output range of ±2.5 V; this is 5# (LONG).
- NUM_STEPS – The number of discrete steps possible with the A/D or D/A converter. For the DT9841, DT9841E, and DT9841-VIB, this number is 2^{24}. For the DT9842/2 and DT9842/8, this number is 2^{16}.
- COUNTS_PER_STEP – The number of count changes for one step of the A/D converter. For the DT9841, DT9841E, and DT9841-VIB, this number is 2^{8}. For the DT9842/2 and DT9842/8, this number is 2^{16}.
- BIP10_COUNTS_PER_VOLT – The number of counts per volt for the ±10 V range of the specified module. This is defined as follows: (NUM_STEPS * COUNTS_PER_STEP/ BIP10_VOLTS_SPAN).
- BIP2PT5_COUNTS_PER_VOLT – The number of counts per volt for the ±2.5 V range of the specified module. This is defined as follows: (NUM_STEPS *COUNTS_PER_STEP/ BIP2PT5_VOLTS_SPAN).

- BIP10_VOLTS_PER_COUNT – The number of volts per count for the ±10 V range of the specified module. This is defined as follows: (1/BIP10_COUNTS_PER_VOLT).
- BIP2PT5_VOLTS_PER_COUNT – The number of volts per count for the ±2.5 V range of the specified module. This is defined as follows: (1/BIP2PT5_COUNTS_PER_VOLT).
- BIP10_MAX_VOLTS – The voltage that corresponds to the maximum count value for the ±10 V range of the specified module. This is defined as follows: (10.# – BIP10_VOLTS_PER_COUNT).
- BIP2PT5_MAX_VOLTS – The voltage that corresponds to the maximum count value for the ±2.5 V range of the specified module. This is defined as follows: (2.5 – BIP2PT5_VOLTS_PER_COUNT).
- BIP10_MIN_VOLTS – The voltage that corresponds to the minimum count value for the ±10 V range of the specified module. This is defined as –10#.
- BIP2PT5_MIN_VOLTS – The voltage that corresponds to the minimum count value for the ±2.5 V range of the specified module. This value (–2.5), is available on DT9841, DT9841E, and DT9841-VIB modules.

Register Constants

The following register constants are defined in the DTCommLib.BAS file. These constants are provided if you want to read the registers of the DT9840 Series modules using **DT_BoardReadRegister**, described on page 91, or **DT_BoardReadFrom Memory**, described on page 90, or write to the registers of the DT9840 Series modules using **DT_BoardWriteRegister**, described on page 93, or **DT_BoardWriteToMemory**, described on page 94:

Note: Refer to the *DT9840 Series User's Manual* for more information on these registers.

- DEVICE_REG_BASE – The Hardware Control and Status register of the DT9840 Series module (address &HB0000000).
- HDW_CTRL_REG – The Hardware Control and Status register (DEVICE_REG_BASE).
- HDW_CALIBRATION_REG – A calibration register (DEVICE_REG_BASE + &H4.
- DT9842_CLOCK_REG – The clock divider register for the DT9842/2 and DT9842/8 modules (DEVICE_REG_BASE + &H8).
- USB_CALIBRATION_REG – A USB writable calibration register (DEVICE_REG_BASE + &HC).
- ID_REG – The model ID and FPGA register (DEVICE_REG_BASE + &H1000).
- AD0_IN_REG – The data register for analog input channel 0 (DEVICE_REG_BASE + &H4000).
- AD1_IN_REG – The data register for analog input channel 1 (DEVICE_REG_BASE + &H4004).
- AD2_IN_REG – The data register for analog input channel 2 (DEVICE_REG_BASE + &H4008). Ignore this register for DT9841E modules.

- AD3_IN_REG – The data register for analog input channel 3 (DEVICE_REG_BASE + &H400C). Ignore this register for DT9841E modules.
- AD4_IN_REG – The data register for analog input channel 4 (DEVICE_REG_BASE + &H4010). Ignore this register for DT9841E modules.
- AD5_IN_REG – The data register for analog input channel 5 (DEVICE_REG_BASE + &H4014). Ignore this register for DT9841E modules.
- AD6_IN_REG – The data register for analog input channel 6 (DEVICE_REG_BASE + &H4018). Ignore this register for DT9841E modules.
- AD7_IN_REG – The data register for analog input channel 7 (DEVICE_REG_BASE + &H401C). Ignore this register for DT9841E modules.
- DIG_IN_REG – The input register for digital input ports 0, 1, and 2, and the flag bits (DEVICE_REG_BASE + &H4020). Port 2 is not used for the DT9841-VIB module
- CTR0_IN_REG – The input register for counter/timer 0 (DEVICE_REG_BASE + &H4024).
- CTR1_IN_REG – The input register for counter/timer 1 (DEVICE_REG_BASE + &H4028).
- CTR2_IN_REG – The input register for counter/timer 2 (DEVICE_REG_BASE + &H402C).
- LED_REG – The register for the diagnostic LEDs (DEVICE_REG_BASE + &H4030). The following constants are provided for controlling individual LEDs:
  - LED_D0 – LED CR9 (&H01).
  - LED_D1 – LED CR10 (&H02).
  - LED_D2 – LED CR11 (&H04).
  - LED_D3 – LED CR8 (&H08).
  - LED_D4 – LED CR14 (&H10).
  - LED_D5 – LED CR7 (&H20).
  - LED_D6 – LED CR12 (&H40).
  - LED_D7 – LED CR13 (&H80).
  - LED_ALL – LEDs CR7 to CR14 (&HFF).
  - LED_HBEAT – LED CR7 (&H20).
  - LED_RED – LED CR7 (&H20). This LED turns red.
  - LED_GREEN – LED CR14 (&H10). This LED turns green.
- DAC0_REG – The data register for analog output channel 0 (DEVICE_REG_BASE + &H8000).
- DAC1_REG – The data register for analog output channel 1 (DEVICE_REG_BASE + &H8004).
- DIG_OUT_REG – The digital output register (DEVICE_REG_BASE + &H8008). Port 2 is not used for the DT9841-VIB module.
- CTR0_PERIOD_REG – The period register for counter/timer 0 (DEVICE_REG_BASE + &HC000).

- CTR1_PERIOD_REG – The period register for counter/timer 1 (DEVICE_REG_BASE + &HC004).
- CTR2_PERIOD_REG – The period register for counter/timer 2 (DEVICE_REG_BASE + &HC008).
- CTR0_PULSE_REG – The pulse register for counter/timer 0 (DEVICE_REG_BASE + &HC010).
- CTR1_PULSE_REG – The pulse register for counter/timer 1 (DEVICE_REG_BASE + &HC014).
- CTR2_PULSE_REG – The pulse register for counter/timer 2 (DEVICE_REG_BASE + &HC018).
- CTR0_CTRL_REG – The control register for counter/timer 0 (DEVICE_REG_BASE + &HC020).
- CTR1_CTRL_REG – The control register for counter/timer 1 (DEVICE_REG_BASE + &HC024).
- CTR2_CTRL_REG – The control register for counter/timer 2 (DEVICE_REG_BASE + &HC028).
- CTR_STATUS_REG – The status register for the counter/timers (DEVICE_REG_BASE + &HC030).
- DIG_IN_MASK_REG – The mask register for digital input port 0 (DEVICE_REG_BASE + &H10000). You can program this register to generate an interrupt when any bit of port 0 changes state.
- DIG_PORT0_REG – The register for digital I/O port 0 (DEVICE_REG_BASE + &H10004).
- DIG_PORT1_REG – The register for digital I/O port 1 (DEVICE_REG_BASE + &H10008).
- DIG_PORT2_REG – The register for digital I/O port 2 (DEVICE_REG_BASE + &H1000C). Port 2 is not used for the DT9841-VIB module.
- SB_CTRL_REG – The Control and Address (CE3) register for the Scalable Bus (DEVICE_REG_BASE + &HB0014000).
- SB_XFER_CTRL_REG – The Scalable Bus Transfer Control register (&HB0014004).
- SB_XFER_STATUS_REG – The Scalable Bus Transfer Status register (&HB0014008).
- SB_REG_BASE – The Scalable Bus write register (&HA0000000).
- SB_OUT_REG – The Scalable Bus write register (&HA0000000).
- SB_IN_REG – The Scalable Bus input register (&HA0000004).

Error Constants

The following error constants are defined in the DTCommLib.BAS file:

- DT_COMMON_BASE_ERR – A value of 0.
- DT_HOST_BASE_ERR – A value of 1000.
- DT_9841_BASE_ERR – A value of 2000.
- DT_HOST_COMM_BASE_ERR – A value of 3000.
- DT_9841_COMM_BASE_ERR – A value of 4000.

Counter/Timer Constants

The following register constants are defined in the DTCommLIB.BAS file:

- COUNTER_RESOLUTION – A value of FFFFFFFFh (2^{32} - 1) when using the internal clock source for counter/timer operations.
- COUNTER_CLOCK_FREQ – A value of 18 MHz; the time base for the internal clock for counter/timer operations.
- COUNTER_CLOCK_PERIOD – A value of 55 ns (1/ COUNTER_CLOCK_FREQ).
- CTR0_ONE_SHOT_IN_PROGRESS – A value of &H1.
- CTR1_ONE_SHOT_IN_PROGRESS – A value of &H2.
- CTR2_ONE_SHOT_IN_PROGRESS – A value of &H4.
- CTR0_MEASURE_IN_PROGRESS – A value of &H10.
- CTR1_MEASURE_IN_PROGRESS – A value of &H20.
- CTR2_MEASURE_IN_PROGRESS – A value of &H40.

Flash Memory Constants

The following flash memory constants are defined in the DTCommLib.BAS file:

- USER_DATA_MEM_SIZE – A value of &H10000.
- DATA_FILE_MAGIC_NUMBER – A value of &H9191956

Enumerated Types

The following enumerated types are defined in the DTCommLib.BAS file:

- ADC_RANGE, described on page 170
- AD_DA_CLK_SRC, described on page 170
- CHAN_TYPE, described on page 171
- CLOCK_SRC, described on page 171
- CLOCK_TYPE, described on page 172
- COUNT_MODE, described on page 173

- DAC_FILTER_TYPE, described on page 176
- DAC_RANGE, described on page 177
- DTSTATUS, described on page 178
- EDGE_TYPE, described on page 181
- ERROR_MODE_SELECT, described on page 181
- GATE_TYPE, described on page 182
- MODEL_ID, described on page 183
- POLARITY, described on page 185
- TRIGGER_TYPE, described on page 185

ADC_RANGE Enumerated Type

Definition

```
Public Enum ADC_RANGE
  ADC_BIPOLAR_10_VOLTS=0
End Enum
```

Include File DTCommLib.BAS

Members

Name: ADC_BIPOLAR_10_VOLTS

Description: If you specify this element, the analog input range is ±10 V.

AD_DA_CLK_SRC Enumerated Type

Definition

```
Public Enum AD_DA_CLK_SRC
  AD_DA_CLK_SRC_INTERNAL=0
  AD_DA_CLK_SRC_EXTERNAL=1
  AD_DA_CLK_SRC_SB_MASTER=2
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: AD_DA_CLK_SRC_INTERNAL

Description: If you specify this element, the internal sample clock on the DT9840 Series module is used.

You specify the frequency of the internal clock source using the CLOCK_SETUP structure, described on page 192.

Name: AD_DA_CLK_SRC_EXTERNAL

Description: If you specify this element, the signal that is connected to the Ext Clk BNC on the DT9840 Series module is used as the sample clock.

For the DT9841, DT9841E, and DT9841-VIB modules, use an external clock source with a 50% duty cycle. Conversions start on a high-to-low transition of the external clock signal after a rising edge of the internal calibration signal. The resulting frequency of the external clock input is always equal to the frequency of the external clock signal that you connected to the module divided by 256 (this division is done internally by the module). For example, if you need a sampling frequency of 100 kHz, use an external clock source with a frequency of 25.6 MHz.

Name: AD_DA_CLK_SRC_SB_MASTER

Description: If you specify this element, the sample clock is derived from the master DT9840 Series module through the Scalable Bus. This is the recommended setting for all slave DT9840 Series modules when you want to synchronize the acquisition of data on all modules.

CHAN_TYPE Enumerated Type

Definition

```
Public Enum CHAN_TYPE
  READ_CHAN=0
  WRITE_CHAN=1
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: READ_CHAN

Description: If 0, the channel on the host is opened for reading.

Name: WRITE_CHAN

Description: If 1, the channel on the host is opened for writing.

CLOCK_SRC Enumerated Type

Definition

```
Public Enum CLOCK_SRC
  INTERNAL_18MHZ=0
  EXTERNAL_CLOCK=1
  CASCADED_CLOCK=2
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: INTERNAL_18MHZ

Description: If you specify this element, the internal C/T clock uses a 18 MHz time base. Counter/timer operations start on the rising edge of the clock input signal.

This clock source is not generally used for event counting or up/down counting mode, and is automatically selected if measure mode or continuous measure mode is specified.

Name: EXTERNAL_CLOCK

Description: If you specify this element, an external C/T clock source paces the specified counter/timer channel. Counter/timer operations start on the rising edge of the clock input signal.

This clock source is not generally used for rate generation, one-shot, or repetitive one-shot mode, and is not supported for measure mode or continuous measure mode.

Name: CASCADED_CLOCK

Description: If you specify this element, the clock input signal of the specified counter/timer is connected in software to the clock output signal of the proceeding counter/timer.

For example, if you specify CASCADE_CLOCK for counter/timer 2, the clock input of counter/timer 2 is internally connected to the clock output of counter/timer 1.

Note that you cannot specify CASCADED_CLOCK for counter/timer 0, since there is no lower counter/timer with which to cascade.

The rising edge of the clock input signal is active.

This clock source is not supported for up/down counting, measure mode, or continuous measure mode.

CLOCK_TYPE Enumerated Type

Definition

```
Public Enum CLOCK_TYPE
  AD_DA_CLOCK=0
  AD_CLOCK=1
  DA_CLOCK=2
End Enum
```

Include File DTCommLib.BAS

Elements

Name: AD_DA_CLOCK

Description: If you specify this element, the clock that paces both analog input and analog output operations is selected.

Name: AD_CLOCK

Description: If you specify this element, the clock that paces analog input operations is selected.

Name: DA_CLOCK

Description: If you specify this element, the clock that paces analog output operations is selected.

COUNT_MODE Enumerated Type

Definition

```
Public Enum COUNT_MODE
  STD_COUNTING=0
  ONE_SHOT=1
  REPETITIVE_ONE_SHOT=2
  MEASURE=3
  UP_DOWN=4
  CONTINUOUS_MEASURE=5
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: STD_COUNTING

Description: If you specify this element, the counter/timer performs an event counting or rate generation (pulse output) operation.

For event counting operations, specify an external C/T clock source. For rate generation operations, specify an internal C/T clock source.

Specify either a software gate, normal external gate, or inverted external gate signal to enable the operation.

When the operation is enabled, the counter increments to the value specified by *PulseWidthCount*, described on page 148, and activates the pulse output.

The pulse output stays active until the counter rolls over to 0 (the terminal count). The pulse output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 195.

The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 196. This sequence repeats as long as the counter is enabled by the gate.

You can read the value of the counter at any time using **DT_ReadCTR**.

Name: ONE_SHOT

Description: If you specify this element, the counter/timer outputs a single pulse when the external gate is active.

Description (cont.): Specify an internal C/T clock source and either a normal external gate or inverted external gate type to enable the operation.

When the operation is enabled, the counter begins incrementing. When the counter increments to the value of *PulseWidthCount*, described on page 196, the value of the counter is output.

The output stays active until the counter rolls over to 0 (the terminal count). When the counter reaches the terminal count, the module clears the one-shot trigger enable bit. The output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 195.

The pulse output then stays inactive, and the counter stays disabled, until the one-shot trigger enable bit is set. The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 196.

Name: REPETITIVE_ONE_SHOT

Description: If you specify this element, the counter/timer outputs a single pulse when the specified external gate signal is active.

Specify an internal C/T clock source and either a normal external gate or an inverted external gate.

When the operation is enabled with an external gate, the counter begins incrementing. When the counter increments to the value of *PulseWidthCount*, described on page 196, the value of the counter is output.

The output stays active until the counter rolls over to 0 (the terminal count). The output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 195.

The user output then stays inactive, and the counter stays disabled, until the next active gate occurs. All gates that occur while the counter is incrementing are ignored. The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 196.

Name: MEASURE

Description: If you specify this element, the counter/timer measures the interval of a signal between a selected measurement start edge and a selected measurement stop edge.

Connect the signal that you want to measure to either the clock input pin or the gate input pin of the counter/timer that you want to use. Then, specify the measurement start edge and the measurement stop edge. Refer to page 181 for more information on the start and stop edges.

Description (cont.): When you trigger the operation with **DT_TriggerCTR**, the counter increments from the time it detects the selected start edge until it detects the stop edge. When it detects the selected stop edge, the counter stops counting.

You can determine whether the measure mode operation is complete or not using **DT_ReadCTRStatus**. When the operation is complete, read the value of the counter using **DT_ReadCTR**.

Use the following equations to determine the frequency, period, and pulse width of the signal:

- *Frequency* = 18 MHz/Number of *Counts*
- *Period* = 1/*Frequency*
- Pulse width = Number of Counts/18 MHz

Note: In measure mode, the internal C/T clock is used to calculate the interval of the signal between the specified start and stop edges. The pulse width and period count are automatically set to 0. You can specify the polarity of the output signal during and after the measurement.

Name: UP_DOWN

Description: If you specify this element, the counter/timer increments when the specified external gate is enabled and decrements when the specified external gate is disabled. The counter/timer increments or decrements on the rising edge of the external clock input signal.

Use UP_DOWN mode only when the position of the object being monitored stays within the range of the counter, as the operation is not be reliable if the counter increments above FFFFFFFh or decrements below 0.

Specify an external C/T clock source; this mode does not support a cascaded C/T source. Specify either a normal external gate, or inverted external gate signal.

When the operation is enabled, the counter increments to the value specified by *PulseWidthCount*, described on page 196, and activates the pulse output.

The pulse output stays active until the counter rolls over to 0 (the terminal count). The pulse output is then deactivated and the counter is automatically reloaded with the initial count specified by *PeriodCount*, described on page 195.

Description (cont.): The active polarity of the pulse output pulse is determined by *OutputPolarity*, described on page 196. This sequence repeats as long as the counter is enabled by the gate.

You can read the value of the counter at any time using **DT_ReadCTR**.

Name: CONTINUOUS_MEASURE

Description: If you specify this element, the counter starts incrementing when it detects a specified start edge and stops incrementing when it detects the next start edge (the stop edge is ignored in this mode). When the operation completes, the counter remains idle until it is next read. On the next read, the current value of the counter (from the previous edge-to-edge measurement operation) is returned and the next edge-to-edge measurement operation is started automatically.

Connect the signal that you want to measure to either the clock input pin or the gate input pin of the counter/timer that you want to use. Then, specify the measurement start edge. Refer to page 181 for more information on the start edge.

Read the value of the counter using **DT_ReadCTR**. If you read the counter before the measurement is complete, 0 is returned.

Note: In continuous measure mode, the internal C/T clock is used to calculate the interval of the signal between the specified start and stop edges. The pulse width and period count are automatically set to 0.

You can use the following equations to determine the frequency and period of the signal based on the value of the counter:

- *Frequency* = 18 MHz/Number of *Counts*
- *Period* = 1/*Frequency*

DAC_FILTER_TYPE Enumerated Type

Definition

```
Public Enum DAC_FILTER_TYPE
  NONE=0
  FIVE_KHZ=1
  TWENTY_KHZ=2
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: NONE

Description: If you specify this element, no filter will be applied to the analog output channels. Use this setting for DT9842/2 and DT9842/8 modules.

Name: FIVE_KHZ

Description: If you specify this element, a 5 kHz reconstruction filter is applied to each analog output channel on the DT9841, DT9841E, and DT9841-VIB. This setting is recommended for slow clock rates.

This element is ignored for the DT9842 and DT9842/8 modules.

Name: TWENTY_KHZ

Description: If you specify this element, a 20 kHz reconstruction filter is applied to each analog output channel on the DT9841, DT9841E, and DT9841-VIB. This setting is recommended for fast clock rates.

This element is ignored for the DT9842 and DT9842/8 modules.

DAC_RANGE Enumerated Type

Definition

```
Public Enum DAC_RANGE
  DAC_BIPOLAR_10_VOLTS = 0
  DAC_BIPOLAR_2_PT_5_VOLTS = 1
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: DAC_BIPOLAR_10_VOLTS

Description: If you specify this element, the analog output range is ±10 V.

Name: DAC_BIPOLAR_2_PT_5_VOLTS

Description: If you specify this element, the analog output range is ±2.5 V.

DTSTATUS Enumerated Type

Definition

```
Public Enum DTSTATUS
  DT_SUCCESS = DT_COMMON_BASE_ERR
   ' 0
  DT_NO_MEMORY
  DT_BUFFER_TOO_SMALL
  DT_INVALID_CHAN_NAME
  DT_INVALID_CHAN_HANDLE
  DT_INVALID_PHCHAN
  DT_NOT_IMPLEMENTED
  DT_INVALID_CALLBACK_PTR
  DT_INVALID_BUFFER_PTR
  DT_INVALID_NUM_BYTES
  DT_INVALID_BYTES_READ_PTR '10
  DT_TIMEOUT
  DT_INVALID_CHAN_TYPE
  DT_INVALID_POINTER
  DT_CALLBACK_NOT_FOUND
  DT_CHAN_DISCONNECTED
  DT_INCOMPATIBLE_CHAN_TYPE
  DT_CHAN_ALREADY_OPEN
  DT_CHAN_ALREADY_CLOSED
   DT_TOO_BIG_FOR_CHAN_BUFFER
   DT_TOO_BIG_FOR_BOARD_BUFFER '20
  DT_VOLTAGE_OUT_OF_RANGE
  DT_AD_VALUE_OUT_OF_RANGE
  DT_NO_SYNC_DATA_RECEIVED
  DT_INVALID_DSP_BUFFER_SIZE_PTR
  DT_INVALID_HOST_BUFFER_SIZE_PTR
  DT_INVALID_BUF_SIZE
  DT_CHAN_OPENED_FOR_WRITE
  DT_CHAN_OPENED_FOR_READ
' Errors that are specific to the host
  DT_INVALID_BOARD_NAME = DT_HOST_BASE_ERR
' 1000
  DT_INVALID_BOARD_HANDLE
  DT_INVALID_PHBOARD
  DT_INVALID_FILENAME
  DT_INVALID_BOARD_INDEX
  DT_INVALID_BOARD_INFO
  DT_INVALID_WINDOW_HANDLE
  DT_FAILURE
  DT_INVALID_MSG_PTR
  DT_INVALID_MSG_BUF_LEN
  DT_INVALID_DSP_ADDRESS
  DT_CANT_OPEN_DRIVER
  DT_CANT_FIND_BOARD
  DT_NO_BOARD_FOR_INDEX
  DT_NO_CHAN_FOR_INDEX
  DT_NO_MESSAGING
```

Definition (cont.)

```
  DT_WINDOW_NOT_FOUND
  DT_FILE_NOT_FOUND
  DT_DRIVER_VERSION
  DT_FIRMWARE_VERSION
  DT_DSP_VERSION
  DT_NO_PROGRAM_LOADED
  DT_ERROR_FILE_READ
  DT_ERROR_INVALID_COFF
  DT_INVALID_BUFFER_ADDRESS
  DT_ERROR_MEMORY_MISMATCH
  DT_USB_ERROR
  DT_INVALID_DSP_ADDRESS_PTR
  DT_INVALID_REGISTER
  DT_BOARD_ALREADY_OPEN
  DT_INVALID_BOARD_OPTIONS
  DT_BOARD_NOT_RESPONDING
  DT_BOARD_WAS_SHUTDOWN
  DT_PROGRAM_TOO_LARGE
' Errors that are specific to the DT9841
  DT_GENERAL_DT9841_ERROR = DT_9841_BASE_ERR
' 2000
  DT_INVALID_SAMPLE_RATE
  DT_NO_DIGITAL_OUT_PORTS
  DT_NO_DIGITAL_IN_PORTS
  DT_IN_LOOP_IS_RUNNING
  DT_OUT_LOOP_IS_RUNNING
  DT_NO_LOOP_RUNNING
  DT_DSP_BIOS_ERROR
  DT_NO_MORE_CHANNELS
  DT_INVALID_VALUE
  DT_INVALID_BLOCK
  DT_INVAID_AD_CHAN
  DT_INVALID_SCAN
  DT_NO_OUTPUTS_ENABLED
  DT_INVALID_COUNTER
  DT_INVALID_SETUP_PTR
  DT_INVALID_MASK
  DT_HANDLER_ALREADY_INSTALLED
  DT_BIT_IS_OUTPUT
  DT_INVALID_COMPARE_VALUE
  DT_COMPARE_IS_OUTPUT
  DT_MASK_IS_OUTPUT
  DT_INVALID_DA_CHAN
  DT_BIT_IS_INPUT
  DT_CLOCK_NOT_RUNNING
  DT_NOT_SB_MASTER
  DT_NOT_SB_SLAVE
```

Definition (cont.)

```
  DT_SB_NOT_ENABLED
  DT_INVALID_CLOCK_SRC
  DT_INVALID_COUNT_MODE
  DT_MEASURE_IN_PROGRESS
  DT_ONESHOT_IN_PROGRESS
  DT_SB_NOT_READY
  DT_NO_DAC_FILTER
  DT_SB_IS_BUSY
  DT_SB_TIMEOUT
  DT_INVALID_DMA_BUFFER
  DT_NO_DMA_CHAN
  DT_ERROR_FILTER
  DT_MUST_INIT_LIBRARY
  DT_RANGE_NOT_SUPPORTED
  DT_NO_SB_MODULE_FOR_INDEX
  DT_ADC_OVERRUN
  DT_DAC_UNDERRUN
  DT_NOT_IEPE_BOARD
  DT_FLASH_WRITE_FAILED
  DT_FLASH_VERIFY_FAILED

' Communication errors detected by host
  DT_HOST_CHAN_ALREADY_OPEN =
   DT_HOST_COMM_BASE_ERR
' 3000
  DT_HOST_CHAN_ALREADY_CLOSED
  DT_HOST_BUFFER_EMPTY
  DT_HOST_RCV_BUFFER_FULL
  DT_HOST_MSG_UNKNOWN_TYPE
  DT_HOST_MSG_INVALID_LENGTH
  DT_HOST_CHAN_BUFFER_FULL
  DT_HOST_SEND_BUFFER_FULL
' Communication errors detected by DT9841
'DT_9841_CHAN_ALREADY_OPEN =
   DT_9841_COMM_BASE_ERR
' 4000
  DT_9841_CHAN_ALREADY_CLOSED
  DT_9841_BUFFER_EMPTY
  DT_9841_RCV_BUFFER_FULL
  DT_9841_MSG_UNKNOWN_TYPE
  DT_9841_MSG_INVALID_LENGTH
  DT_9841_CHAN_BUFFER_FULL
  DT_9841_SEND_BUFFER_FULL
End Enum
```

Public Interface DTCommLib.BAS

Members Each member corresponds to an error code. Refer to Appendix A starting on page 107 for more information on error codes.

EDGE_TYPE Enumerated Type

Definition

```
Public Enum EDGE_TYPE
  SELECTED_GATE_RISES=0
  SELECTED_GATE_FALLS=1
  EXT_CLOCK_RISES=2
  EXT_CLOCK_FALLS=3
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: SELECTED_GATE_RISES

Description: If you specify this element, a rising edge (low to high transition) on the external gate input pin starts or stops the measurement operation. The software automatically sets the gate type to EXT_NORMAL_GATE in this mode.

Name: SELECTED_GATE_FALLS

Description: If you specify this element, a falling edge (high to low transition) on the external gate input pin starts or stops the measurement operation. The software automatically sets the gate type to EXT_NORMAL_GATE in this mode.

Name: EXTERNAL_CLOCK_RISES

Description: If you specify this element, a rising edge (low to high transition) on the external C/T clock input pin starts or stops the measurement operation.

Name: EXTERNAL_CLOCK_FALLS

Description: If you specify this element, a falling edge (high to low transition) on the external C/T clock input pin starts or stops the measurement operation.

Notes EDGE_TYPE is used only in measure mode and continuous measure mode.

ERROR_MODE_SELECT Enumerated Type

Definition

```
Public Enum ERROR_MODE_SELECT
  STOP_ON_ERROR=0
  CONTINUE_ON_ERROR=1
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: STOP_ON_ERROR

Description: If you specify this element, the operation stops if the DT9840 Series module detects an error condition.

Name: CONTINUE_ON_ERROR

Description: If you specify this element, the operation continues if the DT9840 Series module detects an error condition.

GATE_TYPE Enumerated Type

Definition

```
Public Enum GATE_TYPE
  GATED_OFF=0
  GATED_ON=1
  EXT_NORMAL_GATE=2
  EXT_INVERTED_GATE=3
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: GATED_OFF

Description: If you specify this element, the gate is disabled.

Name: GATED_ON

Description: If you specify this element, a software command enables the counter/timer operation immediately after execution. This gate type is not supported for up/down counting, measure mode, continuous measure mode, one-shot, or repetitive one-shot operations.

Name: EXT_NORMAL_GATE

Description: For standard counting operations, enables the operation when the external gate signal is high, and disables the operation when the external gate signal is low.

For one-shot and repetitive one-shot operations, enables the operation on the transition from the low level to the high level (rising edge), and disables the operation on the transition from the high level to the low level (falling edge).

For up/down counting mode, increments the counter when the external gate signal is high, and decrements the counter when the external gate signal is low.

For measure mode and continuous measure mode, this value is automatically set to EXT_NORMAL_GATE if you specify the EDGE_TYPE is SELCTED_GATE_RISES or SELECTED_GATE_FALLS.

Name: EXT_INVERTED_GATE

Description: For standard counting operations, enables the operation when the external gate signal is low, and disables the operation when the external gate signal is high.

For one-shot and repetitive one-shot operations, enables the operation on the transition from the high level to the low level (falling edge), and disables the operation on the transition from the low level to the high level (rising edge).

For up/down counting mode, increments the counter when the external gate signal is low, and decrements the counter when the external gate signal is high.

This gate type is not supported for measure mode or continuous measure mode.

MODEL_ID Enumerated Type

Definition

```
typedef enum
{
  DT9841_128_0 = 0,
  DT9841_64_2 = 1,
  DT9842_128_0 = 2,
  DT9842_64_2 = 3,
  DT9848_128_0 = 4,
  DT9848_64_2 = 5,
  DT9841_128_2_6713 = 6,
  DT9842_128_2_6713 = 7,
  DT9848_128_2_6713 = 8,
  DT9841E_128_2_6713 = 9,
  DT9841_128_2_VIB = 10,
  UNKNOWN_MODEL = -1
} MODEL_ID;
```

Include File DTCommLib.BAS

Elements

Name: DT9841_128_0

Description: This element indicates that you have a DT9841-R module with 128 MBytes of SDRAM and 0 MBytes of flash RAM.

Name: DT9841_64_2

Description: This element indicates that you have a DT9841-F module with 64 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9842_128_0

Description: This element indicates that you have a DT9842/2-R module with 128 MBytes of SDRAM and 0 MBytes of flash RAM.

Name: DT9842_64_2

Description: This element indicates that you have a DT9842/2-F module with 64 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9848_128_0

Description: This element indicates that you have a DT9842/8-R module with 128 MBytes of SDRAM and 0 MBytes of flash RAM.

Name: DT9848_64_2

Description: This element indicates that you have a DT9842/8-F module with 64 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9841_128_2_6713

Description: This element indicates that you have a DT9841 module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9842_128_2_6713

Description: This element indicates that you have a DT9842 module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9848_128_2_6713

Description: This element indicates that you have a DT9848 module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9841E_128_2_6713

Description: This element indicates that you have a DT9841E module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: DT9841_128_2_VIB

Description: This element indicates that you have a DT9841-VIB module (that contains the 6713 chip), with 128 MBytes of SDRAM and 2 MBytes of flash RAM.

Name: UNKNOWN_MODEL

Description: This element indicates that you have a newer module type than what is defined in the version of DTCommLib.DLL on your computer. Contact Data Translation for the most up-to-date files.

Notes Non-6713 modules require customized TCF files. If you are using one of these modules and do not have the TCF file you need, contact Technical Support for help.

POLARITY Enumerated Type

Definition

```
Public Enum POLARITY
  ACTIVE_HIGH=0
  ACTIVE_LOW=1
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: ACTIVE_HIGH

Description: If you specify this element, the high portion of the total pulse output period is the active portion of the counter/timer clock output signal on the DT9840 Series module.

Name: ACTIVE_LOW

Description: If you specify this element, the low portion of the total pulse output period is the active portion of the counter/timer clock output signal on the DT9840 Series module.

Notes In measure mode, if POLARITY is ACTIVE_HIGH when you call **DT_TriggerCTR**, the counter output is low while the measurement is in progress and high when the measurement is complete. If POLARITY is ACTIVE_LOW when you call **DT_TriggerCTR**, the counter output is high while the measurement is in progress and low when the measurement is complete.

TRIGGER_TYPE Enumerated Type

Definition

```
Public Enum TRIGGER_TYPE
  IMMEDIATE=0
  EXT_TRIGGER=2
End Enum
```

Public Interface DTCommLib.BAS

Members

Name: IMMEDIATE

Description: If you specify this element, the operation on the DT9840 Series module starts immediately when an input or output operation function is called.

Name: EXT_TRIGGER

Description: If you specify this element, a trigger event occurs when the DT9840 Series module detects a low to high transition on the TTL-level signal connected to the Ext Trig BNC on the module. This trigger asserts EXT_INT4 for processing.

Data Types

The following data types are defined in the DTCommLib.BAS file:

- LONG – A 32-bit integer.
- DOUBLE – A double, which is always 64-bits IEE format.
- BOARD_NAME_MAX_LEN – The maximum number of characters for the module name is 256.
- ERR_STRING_MAX_LEN – The maximum number of characters for the error string is 256.
- *BRD_HANDLE – A pointer of type VOID.
- *FIND_HANDLE – A pointer of type VOID.
- BOARD_NAME[BOARD_NAME_MAX_LEN] – An array of type CHAR that contains 256 elements.
- ERROR_STRING[ERR_STRING_MAX_LEN] – An array of type CHAR that contains 256 elements.
- MAX_BOARD_OPTION_SIZE – The maximum number of elements for the BOARD_OPTIONS structure is 200.
- CHAN_NAME_MAX_LEN – The maximum number of characters for the channel name is 200.
- *CHAN_HANDLE – A pointer of type VOID.
- CHAN_NAME[CHAN_NAME_MAX_LEN] – An array of type CHAR that contains 200 elements.
- ADC_SETUP, described on page 187
- BOARD_OPTIONS, described on page 188
- BRD_INFO, described on page 190
- CLOCK_SETUP, described page 192
- CTR_SETUP, described on page 194
- DAC_SETUP, described on page 198
- DATA_FILE_HDR, described on page 186
- DIO_SETUP, described on page 200
- IEPE_CHAN_INFO, described on page 203
- IEPE_SETUP, described on page 203
- INPUT_SCAN_RCD, described on page 204
- OUTPUT_SCAN_RCD, described on page 205
- TRIGGER_SETUP, described on page 206

ADC_SETUP Data Type

Definition

```
Type ADC_SETUP
  Termination As Long
  ErrorOption As ERROR_MODE_SELECT
  Unused1 As Long
  Unused2 As Long
  Unused3 As Long
  Unused4 As Long
  Unused5 As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: Termination

Description: An unsigned integer variable that enables or disables 1 kΩ bias return termination resistors for the analog input channels on the DT9841 and DT9841E modules. Since the DT9841-VIB, DT9842/2, and DT9842/8 modules have single-ended inputs only, this element is ignored for these modules.

This feature is particularly useful with floating signal sources on the DT9841 and DT9841E. Refer to the *DT9840 Series Getting Started Manual* for more information on wiring analog inputs to use bias return termination resistance.

Setting a bit to 0 disables the termination resistor for the associated analog input channel. The bit values are as follows:

- Bit 1 (0x01) – Corresponds to analog input channel 0.
- Bit 2 (0x02) – Corresponds to analog input channel 1.
- Bit 4 (0x04) – Corresponds to analog input channel 2.
- Bit 8 (0x08) – Corresponds to analog input channel 3.
- Bit 16 (0x10) – Corresponds to analog input channel 4.
- Bit 32 (0x20) – Corresponds to analog input channel 5.
- Bit 64 (0x40) – Corresponds to analog input channel 6.
- Bit 128 (0x80) – Corresponds to analog input channel 7.

Name: ErrorOption

Description: A variable of type ERROR_MODE_SELECT, described on page 181, that specifies whether to continue or stop the operation when an A/D overrun error condition is detected.

Description (cont.): If you are performing an input operation and an output operation simultaneously using **DT_ScanLoop**, **DT_BlockLoop**, or **DT_ListLoop**, the following restrictions apply to *ErrorOption*:

- If *ErrorOption* in the DAC_SETUP structure is set to CONTINUE_ON_ERROR, set *ErrorOption* in the ADC_SETUP structure to CONTINUE_ON_ERROR.

  In this case, if either an A/D overrun or D/A underrun error occurs, the user-defined callback functions for both operations are still invoked.

- If *ErrorOption* in the DAC_SETUP structure is set to STOP_ON_ERROR, set *ErrorOption* in the ADC_SETUP structure to STOP_ON_ERROR.

  In this case, if either an A/D overrun or D/A underrun error occurs, both operations stop and the user-defined callbacks are NOT invoked.

Name: Unused1

Description: An unsigned integer variable that is currently not used; specify 0 for this member.

Name: Unused2

Description: An unsigned integer variable that is currently not used; specify 0 for this member.

Name: Unused3

Description: An unsigned integer variable that is currently not used; specify 0 for this member.

Name: Unused4

Description: An unsigned integer variable that is currently not used; specify 0 for this member.

Name: Unused5

Description: An unsigned integer variable that is currently not used; specify 0 for this member.

BOARD_OPTIONS Data Type

Definition

```
Type BOARD_OPTIONS
  SizeThisStructure As Long
  ReadBufSizeInKb As Long
  WriteBufSizeInKb As Long
  DbgFlag As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: SizeThisStructure

Description: A variable of type Long that specifies the size of the BOARD_OPTIONS data type. This element is provided for future compatibility.

Name: ReadBufSizeInKb

Description: A variable of type Long that specifies the size of the module input buffer, in kilobytes. If you specify 0 for *ReadBufSizeInKb*, the default size (64 kBytes) is used.

At a minimum, the size of the module input buffer must be at least as large as the largest message that the DT9840 Series module will send to the host (using the **DT_ChanSyncWrite** or **DT_ChanAsyncWrite** function in the DSP program on the module).

If the amount of memory used by the host application is not a concern, we recommend that you set the size of the module input buffer to a multiple of the largest message (**DT_ChanSyncWrite** or **DT_ChanAsyncWrite**) that the DT9840 Series module will send to the host.

Name: WriteBufSizeInKb

Description: A variable of type Long that specifies the size of the module output buffer, in kilobytes. If you specify 0 for *WriteBufSizeInKb*, the default size (64 kBytes) is used. At a minimum, the size of the module output buffer must be at least as large as the largest message that the host will send to the DT9840 Series module (using the **DT_ChanSyncWrite** or **DT_ChanAsyncWrite** function in the host application).

If the amount of memory used by the host application is not a concern, we recommend that you set the size of the module output buffer to a multiple of the largest message (**DT_ChanSyncWrite** or **DT_ChanAsyncWrite**) that the host will send to the DT9840 Series module.

Name: DbgFlag

Description: A variable of type Long that is passed from the DT9840 Series Host Communication Library to the DT9840 Series DSP Library and is used to control serial debugging on the module.

The DT9840 Series DSP Library uses the lowest 16-bits of this value. The upper 16-bits is available to your host program to define as you wish. For example, you can use this flag on the host to control whether to print debug messages to the serial port on the DT9840 Series module using the **DT_Printf** function of the DT9840 Series DSP Library.

Description (cont.): You can read the value of this flag in the DSP program on the module using the **DT_GetDebugFlag** function of the DT9840 Series DSP Library. To turn debugging off, specify 0 for this value.

Notes Ensure that the combined size of *ReadBufSizeInKb* and *WriteBufSizeInKb* does not exceed 48 MBytes. In addition, ensure that you have enough host memory and free RAM on the DT9840 Series module to accommodate the sizes specified in *ReadBufSizeInKb* and *WriteBufSizeInKb.*

If you specify NULL for the *BoardOptions* parameter of the **DT_BoardOpen** function, the module input and output buffers are set to their default sizes (64 kBytes).

BRD_INFO Data Type

Definition

```
Type BRD_INFO
  Model As MODEL_ID
  NotUsed As Long
  MB_RAM As Long
  MB_Flash As Long
  ProgramIsRunning As Boolean
  Dummy1 As Integer
  FPGAVersion As Long
  USBPortType As Long
  SBEnabled As Boolean
  Dummy2 As Integer
  SBAddress As Long
  SBTerminated As Boolean
  Dummy3 As Integer
  Reserved(15) As Long
End Type
```

Include File DTCommLib.BAS

Elements

Name: Model

Description: A variable of type MODEL_ID, described on page 183, that represents the module type.

Name: NotUsed

Description: A variable of type Long that is used internally by the system, but is not meaningful to Visual Basic programmers.

Name: MB_RAM

Description: A variable of type Long that represents the size of the SDRAM on the module, in megabytes.

Name: MB_Flash

Description: A variable of type Long that represents the size of the flash on the module, in megabytes.

Name: ProgramIsRunning

Description: A Boolean variable that indicates whether a DSP program is currently running on the specified DT9840 Series module.

If TRUE, a DSP program is currently running on the specified DT9840 Series module. If FALSE, a DSP program is not currently running on the specified DT9840 Series module.

Name: Dummy1

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: FPGAVersion

Description: An unsigned integer variable in which the version of the FPGA code on the module is returned.

Name: USBPortType

Description: An unsigned integer variable in which the version of the USB port that is used is returned.

Name: SBEnabled

Description: A Boolean variable that indicates whether the Scalable Bus is enabled for the module. If TRUE, the Scalable Bus is enabled for the module. If FALSE, the Scalable Bus is not enabled for the module.

Refer to the *DT9840 Series Getting Started Manual* for more information on enabling the Scalable Bus using the DT9840 Series Control Panel applet.

Name: Dummy2

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: SBAddress

Description: An integer variable that contains the Scalable Bus address of the module. If the module is the master, a bus address of 0 is returned. If the module is a slave, a bus address of 1 through 3 is returned.

Refer to the *DT9840 Series Getting Started Manual* for more information on assigning addresses to modules using the DT9840 Series Control Panel applet.

Name: SBTerminated

Description: A Boolean variable that indicates whether the Scalable Bus is terminated for the module. If TRUE, the Scalable Bus is terminated for the module. If FALSE, the Scalable Bus is not terminated for the module.

Refer to the *DT9840 Series Getting Started Manual* for more information on terminating the Scalable Bus using the DT9840 Series Control Panel applet.

Name: Dummy3

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: Reserved

Description: An integer array of 15 elements that is reserved for future use.

CLOCK_SETUP Data Type

Definition

```
Type CLOCK_SETUP
  ClockSource As AD_DA_CLK_SRC
  Dummy1 As Long
  SampleRate As Double
  MaxSampleRate As Double
  MinSampleRate As Double
  ActualSampleRate As Double
  ClockType As CLOCK_TYPE
  Unused1 As Long
  Unused2 As Long
  Unused3 As Long
  Unused4 As Long
  Unused5 As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: ClockSource

Description: A variable of type AD_DA_CLK_SRC, described on page 170, that specifies the clock source for the input/output operation.

Name: Dummy1

Description: A variable of type Long that is used internally by the system for padding. Do not use this variable.

Name: SampleRate

Description: A variable of type Double that specifies the frequency, in Hz, for the internal sample clock. For the DT9841, DT9841E, and DT9841-VIB modules, values range from 200 Hz to 100,000 Hz. For the DT9842/2 and DT9842/8 modules, values range from 0 Hz to 100,000 Hz.

The actual frequency that the module can achieve may be slightly different than the frequency you specified, due to the accuracy of the clock frequency (0.01%). The actual clock frequency is returned in *ActualSampleRate*.

For the DT9841, DT9841E, and DT9841-VIB modules, the module multiplies the value that you specify for the internal clock frequency by 512 to set the oscillator on the module; the resulting signal from the oscillator is then divided by 2 to provide a clock signal to the A/D and D/A converters that is oversampled 256 times and has a 50% duty cycle.

For example, if you specify an internal clock frequency of 100 kHz, internally the DT9841, DT9841E, and DT9841-VIB modules set the oscillator to 51.2 MHz then divides the resulting signal by 2 to provide a 25.6 MHz signal with a 50% duty cycle to the A/D and D/A converters.

Name: MaxSampleRate

Description: A variable of type Double that indicates the maximum frequency, in Hz, of the internal sample clock. This value is returned by **DT_SetupClock**, described in the *DT9840 Series DSP Library User's Manual*. For all DT9840 Series modules, this value is 100,000 Hz.

Name: MinSampleRate

Description: A variable of type Double that indicates the minimum frequency, in Hz, of the internal sample clock. This value is returned by **DT_SetupClock**, described in the *DT9840 Series DSP Library User's Manual*. For the DT9841, DT9841E, and DT9841-VIB, this value is 200 Hz. For the DT9842/2 and DT9842/8, this value is 0 Hz.

Name: ActualSampleRate

Description: A variable of type Double that indicates the actual sample frequency, in Hz, that the DT9840 Series module could achieve. This value is returned by **DT_SetupClock**, described in the *DT9840 Series DSP Library User's Manual*.

Name: ClockType

Description: A variable of type CLOCK_TYPE, described on page 172, that indicates the clock to use.

Name: Unused1

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused2

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused3

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused4

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused5

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

CTR_SETUP Data Type

Definition

```
Type CTR_SETUP
  CountMode As COUNT_MODE
  ClkSrc As CLOCK_SRC
  PeriodCount As Long
  PulseWidthCount As Long
  OutputPolarity As POLARITY
  GateType As GATE_TYPE
  MeasureModeStartEdge As EDGE_TYPE
  MeasureModeStopEdge As EDGE_TYPE
  Unused1 As Long
  Unused2 As Long
  Unused3 As Long
  Unused4 As Long
  Unused5 As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: CountMode

Description: A variable of type COUNT_MODE, described on page 173, that defines the operation of the specified counter/timer.

Name: ClkSrc

Description: A variable of type CLOCK_SRC, described on page 171, that defines the clock input source for the counter/timer.

In measure mode and continuous measure mode, the software automatically sets this value to INTERNAL_18MHz.

Name: PeriodCount

Description: A variable of type Long that defines the initial value that is loaded into the specified counter/timer. Values range from 1 to FFFFFFFFh. The counter/timer counts from *PeriodCount* to FFFFFFFFh, rolls over to 0 (the terminal count), and then reloads *PeriodCount*.

In event counting operations, this variable determines the value at which to start counting.

In pulse output operations, this variable determines the frequency of the pulse output signal.

This variable is automatically set to 0 if measure mode or continuous measure mode is specified.

Use the following equations to determine the value for PeriodCount when performing pulse output operations; refer to page 169 for more information on the constants used in these equations:

- *PeriodCount* = COUNTER_RESOLUTION + 2 – (COUNTER_CLOCK_FREQ / output frequency)

  For example, if the clock input frequency is 18 MHz and the desired pulse output frequency is 6.0 MHz, the required *PeriodCount* is FFFFFFFEh.

- #clks per output period = COUNTER_RESOLUTION + 2 – *PeriodCount*

  For example, if *PeriodCount* = FFFFFFFEh, three clocks occur for each output period. In this example, the counter counts FFFFFFFEh, FFFFFFFFh, 0, FFFFFFFEh, FFFFFFFFh, 0, and so on.

- output period = (# clks per output period) x COUNTER_CLOCK_PERIOD

  For example, if three clocks occur for each output period and the clock period is 55.555 ns, then the output period is 166.666 ns.

- output frequency = 1 / output period

  For example, if the output period is 166.666 ns, then the output frequency is 6.0 MHz.

  The maximum output frequency is 9 MHz. The minimum output frequency is 0.0042 Hz.

Name: PulseWidthCount

Description: A variable of type Long that determines when the output pulse of the counter/timer is activated; this variable determines the duty cycle of the pulse output signal. Values range from 1 to FFFFFFFFh.

In addition to rate generation, one-shot, and repetitive one-shot operations, you can use this value to determine when to stop counting in event counting applications.

This variable is automatically set to 0 if measure mode or continuous measure mode is specified.

Use the following equations to determine the value for *PulseWidthCount*; refer to page 169 for more information on the constants used in these equations:

- *PulseWidthCount* = COUNTER_RESOLUTION + 1 – (duty cycle x (clock frequency /output frequency))

  For example, if the clock input frequency is 18 MHz, the frequency of the output pulse is 6.0 MHz, and the desired duty cycle is .66 (66%), then the required *PulseWidthCount* is FFFFFFFEh.

- #clks per output pulse = COUNTER_RESOLUTION + 1 – *PulseWidthCount*

  For example, if *PulseWidthCount* = FFFFFFFEh, two clocks occur for each output pulse. In this example, the output is active for counts FFFFFFFFh and 0.

- output pulse width = (#clks per output pulse) x COUNTER_CLOCK_PERIOD

  For example, if the clock period is 55.55 ns and two clock occur for each output pulse, then the output pulse width is 111.11 ns.

- output duty cycle = (output pulse width) / (output period)

  For example, if the output pulse width is 222.2 ns and the output pulse period is 333.3 ns, then the output duty cycle is .66 (66%).

Name: OutputPolarity

Description: A variable of type POLARITY, described on page 185, that defines the polarity of the output pulse for the specified counter/timer.

Description (cont.): In measure mode, if POLARITY is ACTIVE_HIGH when you call **DT_TriggerCTR**, the counter output is low while the measurement is in progress and high when the measurement is complete. If POLARITY is ACTIVE_LOW when you call **DT_TriggerCTR**, the counter output is high while the measurement is in progress and low when the measurement is complete.

Name: GateType

Description: A variable of type GATE_TYPE, described on page 182, that defines the type of gate to use for the specified counter/timer.

For measure mode and continuous measure mode, this value is automatically set to EXT_NORMAL_GATE if you specify the EDGE_TYPE is SELCTED_GATE_RISES or SELECTED_GATE_FALLS.

Name: MeasureModeStartEdge

Description: Used for measure mode operations and continuous measure mode operations only, a variable of type EDGE_TYPE, described on page 181, that specifies what edge of the signal connected to the external gate or external clock input pin starts the measurement.

In continuous measure mode, the counter increments from the first start edge to the next start edge; the stop edge is ignored.

This variable is ignored for all other counter/timer operations.

Name: MeasureModeStopEdge

Description: Used for measure operations only, a variable of type EDGE_TYPE, described on page 181, that specifies what edge of the signal connected to the external gate or external clock input pin stops the measurement. This variable is ignored for all other counter/timer operations.

Name: Unused1

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused2

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused3

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused4

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused5

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

DAC_SETUP Data Type

Definition

```
Type DAC_SETUP
  FilterType As DAC_FILTER_TYPE
  DAInitialValue(NUM_DAC_CHANS-1) As Long
  ErrorOption As ERROR_MODE_SELECT
  DacRange As DAC_RANGE
  Unused1 As Long
  Unused2 As Long
  Unused3 As Long
  Unused4 As Long
  Unused5 As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: FilterType

Description: A variable of type DAC_FILTER_TYPE, described on page 176, that specifies the reconstruction filter to apply to the analog output channels on the DT9840 Series module.

Name: DAInitialValue

Description: For clocked analog output operations, an array of type Long that specifies the initial value to output to the analog output channels when **DT_SetupDAC**, described in the *DT9840 Series DSP Library User's Manual*, is called.

The size of the array is specified by NUM_DAC_CHANS, described on page 164, minus 1.

Name: ErrorOption

Description: A variable of type ERROR_MODE_SELECT, described on page 181, that specifies whether to continue or stop the operation when a D/A underrun error condition is detected.

If you are performing an output operation and an input operation simultaneously using **DT_ScanLoop**, **DT_BlockLoop**, or **DT_ListLoop**, the following restrictions apply to *ErrorOption*:

- If *ErrorOption* in the ADC_SETUP structure is set to CONTINUE_ON_ERROR, set *ErrorOption* in the DAC_SETUP structure to CONTINUE_ON_ERROR.

Description (cont.): In this case, if either a D/A underrun or A/D overrun error occurs, the user-defined callback functions for both operations are still invoked.

- If *ErrorOption* in the ADC_SETUP structure is set to STOP_ON_ERROR, set *ErrorOption* in the DAC_SETUP structure to STOP_ON_ERROR.

  In this case, if either a D/A underrun or A/D overrun error occurs, both operations stop and the user-defined callbacks are NOT invoked.

Name: DacRange

Description: A variable of type DAC_RANGE, described on page 177, that specifies the output range for the analog output channels.

Name: Unused1

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused2

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused3

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused4

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused5

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Notes On the DT9842/2, the two analog output channels are 180 degrees out of phase with each other. On the DT9842/8, each analog output channel is 45 degrees out of phase with the previous channel.

To convert a voltage to a raw initial value, use **DT_VoltsToDAValue**, described on page 106.

We recommend that you turn on the 20 kHz filter whenever you want the smoothest possible analog output signal; this is particularly true at clock frequencies lower than 50 kHz.

DATA_FILE_HDR

Definition

```
Type DATA_FILE_HDR
  DataFileID As Long
  NumBytes As Long
End Type
```

Include File DTCommLib.BAS

Elements

Name: DataFileID

Description: A variable of type Long that identifies the data file. This variable must be set to the value of DATA_FILE_MAGIC_NUMBER, defined on page 169.

Name: NumBytes

Description: A variable of type Long that specifies the number of bytes to write to or read from the 64 KB reserved block of flash memory. This value should be greater than 0 and less than USER_DATA_MEM_SIZE, defined on page 169.

Notes This data type is used to define the header of each data file to write to or read from flash memory using the DT9840 Series Flash Programmer Utility. The header is followed by the actual data to write to or read from the 64 KB reserved block in flash memory.

DIO_SETUP Data Type

Definition

```
Type DIO_SETUP
  ClockDout As Boolean
  Dummy1 As Integer
  Port0Out As Boolean
  Dummy2 As Integer
  Port1Out As Boolean
  Dummy3 As Integer
  Port2Out As Boolean
  Dummy4 As Integer
  DeglitchPort0 As Boolean
  Dummy5 As Integer
  InitialDOValue As Long
  Dummy6 As Integer
  Unused1 As Long
  Unused2 As Long
  Unused3 As Long
  Unused4 As Long
  Unused5 As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: ClockDout

Description: A Boolean value that specifies whether to clock digital output values while analog output values are being clocked.

If TRUE, digital output values are output with analog output values at the frequency of the sample clock when any of the following functions in the DT9840 Series DSP Library is called:

- DT_ScanLoop
- DT_BlockLoop
- DT_FunctionGen
- DT_ListLoop
- DT_WriteDACDIO

If FALSE, digital output values are not clocked out when analog output values are clocked out.

Name: Dummy1

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: Port0Out

Description: A Boolean variable that specifies whether digital port 0 is an input or an output port. If FALSE, digital port 0 is an input port; if TRUE, digital port 0 is an output port.

Name: Dummy2

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: Port1Out

Description: A Boolean variable that specifies whether digital port 1 is an input or an output port. If FALSE, digital port 1 is an input port; if TRUE, digital port 1 is an output port.

Name: Dummy3

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: Port2Out

Description: A Boolean variable that specifies whether digital port 2 is an input or an output port. If FALSE, digital port 2 is an input port; if TRUE, digital port 2 is an output port.

Name: Dummy4

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: DeglitchPort0

Description: A Boolean variable that specifies whether to enable or disable deglitch circuitry on digital port 0; this function minimizes false triggers due to noise and is particularly useful when using the interrupt-on-change feature.

If TRUE, a 10 ms deglitch function is enabled for digital port 0. If FALSE, the 10 ms deglitch function is disabled for digital port 0.

This parameter is ignored if *Port0Out* is TRUE.

Name: Dummy5

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: InitialDOValue

Description: A variable of type Long that specifies the initial value to output to the digital output lines when **DT_SetupDIO**, described in the *DT9840 Series DSP Library User's Manual,* is called.

Name: Dummy6

Description: A variable of type Integer that is used internally by the system for padding. Do not use this variable.

Name: Unused1

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused2

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused3

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused4

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

Name: Unused5

Description: An unsigned integer variable that is currently unused; specify 0 for this member.

IEPE_CHAN_INFO DataType

Definition

```
Type IEPE_CHAN_INFO
  CurrentSourceEnabled As Boolean
  FilterEnabled As Boolean
  ACCoupling As Boolean
End Type
```

Include File DTCommLib.BAS

Elements

Name: CurrentSourceEnabled

Description: A Boolean variable that indicates whether the 4 mA current source is enabled or disabled for the analog input channel.

If the value is TRUE, the 4 mA current source is enabled. If the value is FALSE, the 4 mA current source is disabled.

Name: FilterEnabled

Description: A Boolean variable that indicates whether the 10 kHz filter is enabled or disabled for the analog input channel.

If the value is TRUE, the 10 kHz filter is enabled. If the value is FALSE, the 10 kHz filter is disabled.

Name: ACCoupling

Description: A Boolean variable that indicates whether AC coupling is enabled or disabled for the analog input channel.

If the value is TRUE, AC coupling is enabled. If the value is FALSE, AC coupling is disabled, and DC coupling is used.

Notes This data type is used by the IEPE_SETUP data type, described on page 154, and applies to the DT9841-VIB module only.

IEPE_SETUP DataType

Definition

```
Type IEPE_SETUP
  Chan(NUM_ADC_REGS - 1)
   As IEPE_CHAN_INFO
End Type
```

Include File DTCommLib.BAS

Elements

Name: Chan[NUM_ADC_REGS]

Description: An array of type IEPE_CHAN_INFO, described on page 203, that specifies the IEPE parameters for all of the analog input channels on the DT9841-VIB module.

Description (cont.): The array is defined to be as large as the number of analog input registers. Refer to page 164 or more information on NUM_ADC_REGS.

Notes This data type applies to the DT9841-VIB module only.

INPUT_SCAN_RCD Data Type

Definition

```
Type INPUT_SCAN_RCD
  ADCValues(NUM_ADC_REGS - 1) As Long
  FlagsAndDin As Long
  CounterValues(NUM_COUNTERS - 1) As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: ADCValues[NUM_ADC_REGS]

Description: A Long array in which the acquired value of all the analog input channels is returned.

The array is defined to be as large as the number of analog input registers. Refer to page 164 for more information on NUM_ADC_REGS.

Name: FlagsandDin

Description: A variable of type Long in which the state of the digital input ports and the eight hardware flags is returned.

Bits 0 to 7 represent port 0, bits 8 to 15 represent port 1, bits 9 to 23 represent port 2, and bits 24 to 31 represent the following hardware flags:

- Bit 24

  1 = Temperature is 65° C or over.
  0 = Temperature is under 65° C.

- Bit 25

  1 = Counter/timer 2 has overflowed.
  0 = Counter/timer 2 has not overflowed.

- Bit 26

  1 = Counter/timer 1 has overflowed.
  0 = Counter/timer 1 has not overflowed.

- Bit 27

  1 = Counter/timer 0 has overflowed.
  0 = Counter/timer 0 has not overflowed.

Description (cont.):

- Bit 28

  1 = A FIFO Full error occurred on the Scalable Bus.
  0 = A FIFO Full error did not occur on the Scalable Bus.

- Bit 29

- 1 = An analog output trigger occurred.
  0 = An analog output trigger did not occur.

- Bit 30

  1 = An analog input trigger error occurred.
  0 = An analog input trigger error did not occur.

- Bit 31

  1 = An analog input trigger occurred.
  0 = An analog input trigger did not occur.

Name: CounterValues[NUM_COUNTERS]

Description: A LONG array in which the acquired value of all the counter/timer channels is returned.

The array is defined to be as large as the number of counter/timer channels. The DT9840 Series modules support three counter/timer channels (0 to 2).

OUTPUT_SCAN_RCD Data Type

Definition

```
Type OUTPUT_SCAN_RCD
  DACValues(NUM_DAC_CHANS - 1) As Long
  DoutValues As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: DACValues(NUM_DAC_CHANS - 1)

Description: An Long array to output to the analog output channels.

The array is defined to be as large as the number of analog output channels on the module.

Name: DoutValues

Description: A variable of type Long that contains a 24-bit value to output to the digital output lines. Bits 0 to 7 represent port 0, bits 8 to 15 represent port 1, and bits 9 to 23 represent port 2.

Notes You must specify this elements of this structure appropriately before calling **DT_ScanLoop**, **DT_BlockLoop**, **DT_ListLoop**, or **DT_FunctionGen**, described in the *DT9840 Series DSP Library User's Manual*.

TRIGGER_SETUP Data Type

Definition

```
Type TRIGGER_SETUP
  TrigType As TRIGGER_TYPE
  Unused1 As Long
  Unused2 As Long
End Type
```

Public Interface DTCommLib.BAS

Members

Name: TrigType

Description: A variable of type TRIGGER_TYPE, described on page 185, that specifies the trigger mode.

Name: Unused1

Description: An unsigned integer variable that is currently used; specify 0 for this member.

Name: Unused2

Description: An unsigned integer variable that is currently used; specify 0 for this member.

Communication Events and Messages

DTEVENT_AD_OVERRUN 208
DTEVENT_ASYNC_DATA_RECEIVED 208
DTEVENT_CHAN_IN_BUFFER_FULL 209
DTEVENT_DA_UNDERRUN 210
DTEVENT_ERROR 210
DTEVENT_INPUT_SCAN_BLOCK_OVERRUN 211
DTEVENT_MODULE_PLUGGED 211
DTEVENT_MODULE_UNPLUGGED 212
DTEVENT_OUT_BUFFER_SPACE_AVAILABLE 213
DTEVENT_OUTPUT_SCAN_BLOCK_UNDERRUN 214
DTEVENT_REMOTE_CHAN_CLOSED 214
DTEVENT_REMOTE_CHAN_OPENED 215
DTEVENT_SYNC_DATA_RECEIVED 215

DTEVENT_AD_OVERRUN

Syntax

```
DTEVENT_AD_OVERRUN(
   Param1,
   Param2
   )
```

Description Indicates that an A/D overrun error occurred.

Parameters

Name: Param1

Description: This parameter is not used (NULL).

Name: Param2

Description: This parameter is not used (NULL).

Notes You can use this event with **DT_RegisterCallback** in your DSP program to detect A/D overrun errors. This event never occurs on the host system.

If this event occurs in your DSP program, ensure that the host program reads analog input data from the DT9840 Series module as fast as it is being acquired.

If you are running **DT_ScanLoop** at 100 kHz, you may get a DTEVENT_AD_OVERRUN or DTEVENT_DA_UNDERRUN event the first time through the scan loop. This event occurs because the interrupt service routine takes longer to execute the first time through the scan loop. Ignore these events the first time through the loop; subsequent events, however, should be addressed.

DTEVENT_ASYNC_DATA_RECEIVED

Syntax

```
DTEVENT_ASYNC_DATA_RECEIVED(
   Param1,
   Param2
   )
```

Description For an open communication channel, indicates that new data was received asynchronously.

Parameters

Name: Param1

Description: The handle of communication channel that received the new data.

Name: Param2

Description: The number of bytes received.

Notes In response to this event, the Windows message DTMSG_ASYNC_DATA_RECEIVED is sent to the host application. The host application should read the data by calling **DT_ChanReadAvailable**, described on page 67, or with **DT_ChanRead**, described on page 65.

DTEVENT_CHAN_IN_BUFFER_FULL

Syntax

```
DTEVENT_CHAN_IN_BUFFER_FULL(
    Param1,
    Param2
    )
```

Description For a channel that was opened for reading, indicates that the channel input buffer is full. The system cannot process any incoming data on any channel until some data has been read from the full buffer.

Parameters

Name: Param1

Description: The handle of the communication channel whose input buffer is full.

Name: Param2

Description: The number of bytes of available buffer space.

Notes In response to this event, the Windows message DTMSG_CHAN_IN_BUFFER_FULL is sent to the host application. The host application program should read the data from the communication channel as soon as possible by calling **DT_ChanReadAvailable**, described on page 67, or with **DT_ChanRead**, described on page 65.

In most well-designed programs, this event should not occur. In most cases, this event indicates that one or more of the following conditions exists on the host:

- The channel input buffer is too small for the application. Increase the size of the channel input buffer by calling **DT_ChanOpen**, described on page 63, in your host application program.
- The application program was busy with another operation for too long and did not read the data from the communication channel in a timely manner. Reorganize the host application program so that reading data from the channel input buffer is a high priority operation.

Notes (cont.)

- The host application is reading fewer bytes than the DSP program on the module is writing to it. If the host application program is calling **DT_ChanRead** instead of **DT_ChanReadAvailable**, ensure that it is reading as many bytes as are being written by the DT9840 Series module.

DTEVENT_DA_UNDERRUN

Syntax

```
DTEVENT_DA_UNDERRUN(
   Param1,
   Param2
   )
```

Description Indicates that an D/A underrun error occurred.

Parameters

Name: Param1

Description: This parameter is not used (NULL).

Name: Param2

Description: This parameter is not used (NULL).

Notes You can use this event with **DT_RegisterCallback** in your DSP program to detect D/A overrun errors. This event never occurs on the host system.

If this event occurs in your DSP program, ensure that the host computer provides analog output data to the DT9840 Series module faster than the DACs are converting the data.

If you are running **DT_ScanLoop** at 100 kHz, you may get a DTEVENT_AD_OVERRUN or DTEVENT_DA_UNDERRUN event the first time through the scan loop. This event occurs because the interrupt service routine takes longer to execute the first time through the scan loop. Ignore these events the first time through the loop; subsequent events, however, should be addressed.

DTEVENT_ERROR

Syntax

```
DTEVENT_ERROR(
      Param1,
      Param2
      )
```

Description Indicates that a synchronous error (such as a **DT_ChanSyncWrite** failure) occurred in the host application, or an asynchronous communication error (background task failure) occurred in the host application.

Parameters

Name: Param1

Description: The handle of the module that generated the error.

Name: Param2

Description: The error code.

Notes Currently, this event occurs only on the host.

In response to this event, the Windows message DTMSG_ERROR is sent to the host application.

DTEVENT_INPUT_SCAN_BLOCK_OVERRUN

Syntax

```
DTEVENT_INPUT_SCAN_BLOCK_OVERRUN(
   Param1,
   Param2
   )
```

Description Indicates that an A/D overrun error occurred in an block or list loop operation.

Parameters

Name: Param1

Description: This parameter is not used (NULL).

Name: Param2

Description: This parameter is not used (NULL).

Notes You can use this event with **DT_RegisterCallback** in your DSP program to detect A/D overrun errors in block loop or list loop operations. This event never occurs on the host system.

If this event occurs in your DSP program, ensure that the host program reads analog input data from the DT9840 Series module as fast as it is being acquired.

DTEVENT_MODULE_PLUGGED

Syntax

```
DTEVENT_MODULE_PLUGGED(
     Param1,
     Param2
     )
```

Description Indicates that the USB cable between the host computer and the DT9840 Series module was plugged in after being unplugged.

Parameters

Name: Param1

Description: This parameter is not used (NULL).

Name: Param2

Description: This parameter is not used (NULL).

Notes This event occurs on the DSP only. This event is generated each time you run the COFF file.

DTEVENT_MODULE_UNPLUGGED

Syntax

```
DTEVENT_MODULE_UNPLUGGED(
      Param1,
      Param2
      )
```

Description Indicates that the USB cable between the host computer and the DT9840 Series module was unplugged.

Parameters

Name: Param1

Description: The handle of the module that generated the error.

Name: Param2

Description: This parameter is not used (NULL).

Notes This event can occur on either the host or DSP. In response to this event on the host, the Windows message DTMSG_MODULE_UNPLUGGED is sent to the host application.

When this event occurs, the host should close all its communication channels, and then close the DT9840 Series module with **DT_BoardClose** (the host application does not have to be closed). The DSP program should then respond by closing its communication channels.

Note: It is imperative that the host application calls **DT_BoardClose** when the DTEVENT_MODULE_UNPLUGGED event occurs to prevent board open failures.

Notes (cont.) To reestablish communication after the USB cable is disconnected, do the following:

1. Reconnect the USB cable.
2. Call **DT_BoardOpen** from the host application. If you want to reestablish communication with a running DSP program, call **DT_BoardGetInfo** to verify that the DSP program is running.
3. Open the communication channels from the host application.
4. Open the communication channels from the DSP program.
5. Download a COFF file to the DT9840 Series module and run it.

DTEVENT_OUT_BUFFER_SPACE_AVAILABLE

Syntax

```
DTEVENT_OUT_BUFFER_SPACE_
   AVAILABLE(
     Param1,
     Param2
     )
```

Description Indicates that a previously full module output buffer was emptied; data was transmitted from the module output buffer across the communication link.

Parameters

Name: Param1

Description: The handle of the communication channel that generated the DT_HOST_RCV_BUFFER_FULL error, described on page 114.

Name: Param2

Description: The number of bytes of available buffer space.

Notes This event is fired on the host only, after a write operation (**DT_ChanAsyncWrite**, described on page 60, or **DT_ChanSyncWrite**, described on page 68) generates a DT_HOST_RCV_BUFFER_FULL error, and data is taken out of the channel input buffer.

In response to this event, the Windows message DTMSG_OUT_BUFFER_SPACE_AVAILABLE is sent to the host application. The host application should retry the write operation.

DTEVENT_OUTPUT_SCAN_BLOCK_UNDERRUN

Syntax

```
DTEVENT_OUTPUT_SCAN_BLOCK_
   UNDERRUN(
     Param1,
     Param2
     )
```

Description Indicates that an D/A underrun error occurred in a block loop or list loop operation.

Parameters

Name: Param1

Description: This parameter is not used (NULL).

Name: Param2

Description: This parameter is not used (NULL).

Notes You can use this event with **DT_RegisterCallback** in your DSP program to detect D/A overrun errors in a block loop or list loop operation. This event never occurs on the host system.

If this event occurs in your DSP program, ensure that the host computer provides analog output data to the DT9840 Series module faster than the DACs are converting the data.

DTEVENT_REMOTE_CHAN_CLOSED

Syntax

```
DTEVENT_REMOTE_CHAN_CLOSED(
   Param1,
   Param2
   )
```

Description Indicates that the DT9840 Series DSP program has closed its side of the communication channel.

Parameters

Name: Param1

Description: The handle of the communication channel that was closed by the DSP program on the DT9840 Series module.

Name: Param2

Description: This parameter is not used (NULL).

Notes In response to this event, the Windows message DTMSG_REMOTE_CHAN_CLOSED is sent to the host application. The host application should close its side of the communication channel by calling **DT_ChanClose**, described on page 61.

DTEVENT_REMOTE_CHAN_OPENED

Syntax

```
DTEVENT_REMOTE_CHAN_OPENED(
    Param1,
    Param2
    )
```

Description Indicates that the DSP program on the DT9840 Series module has opened its side of the communication channel.

Parameters

Name: Param1

Description: The handle of the module whose communication channel was opened by the DT9840 Series module.

Name: Param2

Description: A pointer to a CHAN_OPEN_INFO structure, described on page 145, that contains information about the communication channel on the DT9840 Series module.

Notes In response to this event, the Windows message DTMSG_REMOTE_CHAN_OPENED is sent to the host application. The host application should open its side of the communication channel by calling **DT_ChanOpen**, described on page 63.

DTEVENT_SYNC_DATA_RECEIVED

Syntax

```
DTEVENT_SYNC_DATA_RECEIVED(
    Param1,
    Param2
    )
```

Description For an open communication channel, indicates that new data was received synchronously.

Parameters

Name: Param1

Description: The handle of communication channel that received the new data.

Name: Param2

Description: The number of bytes received.

Notes In response to this event, the Windows message DTMSG_SYNC_DATA_RECEIVED is sent to the host application.I The host application should do the following:

1. Read the data by calling **DT_ChanReadAvailable**, described on page 67, or with **DT_ChanRead**, described on page 65.
2. Process the data.
3. Signal the DSP program on the DT9840 Series module that the data has been processed by calling **DT_ChanAcknowledge**, described on page 59.

Index

Symbols

*PDATA_FILE_HDR structure 151

Numerics

32-bit library 16
64-bit library 16

A

acknowledging a synchronous write request 29
AD_DA_CLK_SRC enumerated type
  in Visual Basic 170
  in Visual C++ 124
ADC_RANGE enumerated type
  in Visual Basic 170
  in Visual C++ 123
ADC_SETUP data type, Visual Basic 187
ADC_SETUP structure, Visual C++ 140
ARM_SELECT enumerated type
  in Visual Basic 185

B

BOARD_OPTIONS data type, Visual Basic 188
BOARD_OPTIONS structure, Visual C++ 141, 143
board-level functions
  DT_BoardClose 11, 24, 38, 41, 49, 85
  DT_BoardDownload 11, 25, 37, 39, 50, 86
  DT_BoardGetInfo 11, 51, 87
  DT_BoardGetName 11, 23, 51, 88
  DT_BoardOpen 11, 23, 37, 39, 52, 89
  DT_BoardReadFromMemory 11, 28, 54, 90
  DT_BoardReadRegister 11, 28, 55, 91
  DT_BoardRun 11, 25, 37, 39, 56, 92
  DT_BoardWriteRegister 11, 30, 57, 93
  DT_BoardWriteToMemory 11, 30, 58, 94
  DT_GetErrorString 11
  DT_RegisterCallback 12, 31, 39, 70
  DT_RegisterMsgHandler 12, 31, 73
  DT_UnregisterCallback 12, 31, 41, 75
  DT_UnregisterMsgHandler 12, 31, 76
BRD_INFO data type, Visual Basic 190
building host programs 16

C

callback functions 162
CALLBACKPROC callback function 162
CHAN_OPEN_INFO structure, Visual C++ 145
CHAN_TYPE enumerated type
  in Visual Basic 171
  in Visual C++ 124
channel-level functions
  DT_ChanAcknowledge 12, 29, 38, 41, 59
  DT_ChanAsyncWrite 12, 29, 37, 40, 60, 95
  DT_ChanBytesAvailable 13, 27, 61, 96
  DT_ChanClose 12, 26, 38, 41, 61, 97
  DT_ChanGetInfo 13, 26, 62, 98
  DT_ChanOpen 12, 26, 37, 39, 63, 99
  DT_ChanRead 13, 27, 38, 41, 65, 101
  DT_ChanReadAvailable 13, 27, 38, 41, 67, 103
  DT_ChanSyncWrite 12, 29, 40, 68
  DT_GetErrorString 34, 69, 104
CLOCK_SETUP data type, Visual Basic 192
CLOCK_SETUP structure, Visual C++ 145
CLOCK_SRC enumerated type
  in Visual Basic 171
  in Visual C++ 125
CLOCK_TYPE enumerated type 126
  in Visual Basic 172
closing a module 24, 26
communication channel information 26
constants
  error 122
  in Visual Basic 164
  in Visual C++ 116
conversion constants
  in Visual Basic 165
  in Visual C++ 118
conversion functions
  DT_ADBlockToVolts 13, 32, 38, 41, 45
  DT_ADChanBlockToVolts 13, 32, 38, 41, 46
  DT_ADScanToVolts 13, 32, 38, 41, 47, 84
  DT_ADValueToVolts 13, 32, 38, 41, 48, 85
  DT_VoltsToDABlock 14, 33, 37, 40, 77
  DT_VoltsToDAChanBlock 14, 33, 37, 40, 78
  DT_VoltsToDAScan 14, 33, 37, 40, 80, 105
  DT_VoltsToDAValue 14, 37, 40, 81, 106
  DT_VoltstoDAValue 33
converting raw analog data to voltage values
  a channel with in a block of data 32

a scan of data 32
a single value 32
blocks of data 32
converting voltage values to raw analog data
a channel with in a block of data 33
a scan of data 33
a single value 33
blocks of data 33
COUNT_MODE enumerated type
in Visual Basic 173
in Visual C++ 126
counter/timer constants
in Visual Basic 169
in Visual C++ 122
CTR_SETUP data type, Visual Basic 194
CTR_SETUP structure, Visual C++ 147

D

DAC_FILTER_TYPE enumerated type
in Visual Basic 176
in Visual C++ 130
DAC_RANGE enumerated type
in Visual Basic 177
in Visual C++ 131
DAC_SETUP data type, Visual Basic 198
DAC_SETUP structure, Visual C++ 150
data types
in Visual Basic 186
in Visual C++ 116
DATA_FILE_HDR data type, Visual Basic 200
DATA_FILE_HDR structure 151
DIO_SETUP data type, Visual Basic 200
DIO_SETUP structure, Visual C++ 152
downloading a DSP program 25
DT_9841_MSG_UNKNOWN_TYPE 114
DT_AD_VALUE_OUT_OF_RANGE 109
DT_ADBlockToVolts 13, 32, 38, 41, 45
DT_ADChanBlockToVolts 13, 32, 38, 41, 46
DT_ADScanToVolts 13, 32, 38, 41
in Visual Basic 84
in Visual C++ 47
DT_ADValueToVolts 13, 32, 38, 41
in Visual Basic 85
in Visual C++ 48
DT_BOARD_ALREADY_OPEN 113
DT_BOARD_NOT_RESPONDING 113
DT_BOARD_WAS_SHUTDOWN 113
DT_BoardClose 11, 24, 38, 41
in Visual Basic 85
in Visual C++ 49
DT_BoardDownload 11, 25, 37, 39
in Visual Basic 86
in Visual C++ 50
DT_BoardGetInfo 11
in Visual Basic 87
in Visual C++ 51
DT_BoardGetName 11, 23
in Visual Basic 88
in Visual C++ 51
DT_BoardOpen 11, 23, 37, 39
in Visual Basic 89
in Visual C++ 52
DT_BoardReadFromMemory 11, 28
in Visual Basic 90
in Visual C++ 54
DT_BoardReadRegister 11, 28
in Visual Basic 91
in Visual C++ 55
DT_BoardRun 11, 25, 37, 39
in Visual Basic 92
in Visual C++ 56
DT_BoardWriteRegister 11, 30
in Visual Basic 93
in Visual C++ 57
DT_BoardWriteToMemory 11, 30
in Visual Basic 94
in Visual C++ 58
DT_BUFFER_TOO_SMALL 108
DT_CALLBACK_NOT_FOUND 108
DT_CANT_FIND_BOARD 111
DT_CANT_OPEN_DRIVER 111
DT_CHAN_ALREADY_CLOSED 109
DT_CHAN_ALREADY_OPEN 109
DT_CHAN_DISCONNECTED 109
DT_CHAN_OPENED_FOR_READ 110
DT_CHAN_OPENED_FOR_WRITE 110
DT_ChanAcknowledge 12, 29, 38, 41, 59
DT_ChanAsyncWrite 12, 29, 37, 40
in Visual Basic 95
in Visual C++ 60
DT_ChanBytesAvailable 13, 27
in Visual Basic 96
in Visual C++ 61
DT_ChanClose 12, 26, 38, 41
in Visual Basic 97
in Visual C++ 61
DT_ChanGetInfo 13, 26
in Visual Basic 98
in Visual C++ 62

DT_ChanOpen 12, 26, 37, 39
in Visual Basic 99
in Visual C++ 63
DT_ChanRead 13, 27, 38, 41
in Visual Basic 101
in Visual C++ 65
DT_ChanReadAvailable 13, 27, 38, 41
in Visual Basic 103
in Visual C++ 67
DT_ChanSyncWrite 12, 29, 40
in Visual C++ 68
DT_DRIVER_VERSION 111
DT_DSP_ VERSION 112
DT_DSP_VERSION 112
DT_ERROR_ MEMORY_MISMATCH 112
DT_ERROR_FILE_READ 112
DT_ERROR_INVALID_COFF 112
DT_FAILURE 110
DT_FILE_NOT_FOUND 111
DT_FIRMWARE_VERSION 112
DT_GetErrorString 11, 34
in Visual Basic 104
in Visual C++ 69
DT_HOST_BUFFER_EMPTY 113
DT_HOST_CHAN_ALREADY_CLOSED 113
DT_HOST_CHAN_ALREADY_OPEN 113
DT_HOST_CHAN_BUFFER_FULL 114
DT_HOST_MSG_INVALID_LENGTH 114
DT_HOST_RCV_BUFFER_FULL 114
DT_HOST_SEND_BUFFER_FULL 114
DT_INCOMPATIBLE_CHAN_TYPE 109
DT_INVALID_ BUFFER_ADDRESS 112
DT_INVALID_ BYTES_READ_PTR 108
DT_INVALID_ CHAN_TYPE 108
DT_INVALID_ PHBOARD 110
DT_INVALID_ PHCHAN 108
DT_INVALID_BOARD_HANDLE 110
DT_INVALID_BOARD_INDEX 110
DT_INVALID_BOARD_INFO 110
DT_INVALID_BOARD_NAME 110
DT_INVALID_BUF_SIZE 110
DT_INVALID_BUFFER_PTR 108
DT_INVALID_CALLBACK_PTR 108
DT_INVALID_CHAN_HANDLE 108
DT_INVALID_CHAN_NAME 108
DT_INVALID_CHAN_TYPE 108
DT_INVALID_DSP_ADDRESS 111
DT_INVALID_DSP_ADDRESS_PTR 113
DT_INVALID_DSP_BUFFER_SIZE_PTR 109
DT_INVALID_FILENAME 110
DT_INVALID_HOST_BUFFER_SIZE_PTR 109
DT_INVALID_MSG_BUF_LEN 110
DT_INVALID_MSG_PTR 110
DT_INVALID_NUM_BYTES 108
DT_INVALID_POINTER 108
DT_INVALID_REGISTER 113
DT_INVALID_WINDOW_HANDLE 110
DT_NO_ PROGRAM_LOADED 112
DT_NO_BOARD_FOR_INDEX 111
DT_NO_CHAN_FOR_INDEX 111
DT_NO_MEMORY 108
DT_NO_MESSAGING 111
DT_NO_SYNC_DATA_RECEIVED 109
DT_NOT_IMPLEMENTED 108
DT_PROGRAM_TOO_LARGE 113
DT_ReadCTR 127, 128, 129, 130, 173, 175, 176
DT_ReadCTRStatus 128, 175
DT_RegisterCallback 12, 31, 39, 70
DT_RegisterMsgHandler 12, 31, 73
DT_SUCCESS 108
DT_TIMEOUT 108
DT_TOO_BIG_FOR_BOARD_BUFFER 109
DT_TOO_BIG_FOR_CHAN_BUFFER 109
DT_TriggerCTR 128, 175
DT_UnregisterCallback 12, 31, 41, 75
DT_UnregisterMsgHandler 12, 31, 76
DT_USB_ERROR 112
DT_VOLTAGE_OUT_OF_RANGE 109
DT_VoltsToDABlock 14, 33, 37, 40, 77
DT_VoltsToDAChanBlock 14, 33, 37, 40, 78
DT_VoltsToDAScan 14, 33, 37, 40
in Visual Basic 105
in Visual C++ 80
DT_VoltsToDAValue 14, 37, 40
in Visual Basic 106
in Visual C++ 81
DT_VoltstoDAValue 33
DT_WINDOW_NOT_FOUND 111
DTEVENT_AD_OVERRUN 208
DTEVENT_ASYNC_DATA_RECEIVED 208
DTEVENT_CHAN_IN_BUFFER_FULL 209
DTEVENT_DA_UNDERRUN 210
DTEVENT_ERROR 210
DTEVENT_INPUT_SCAN_BLOCK_OVERRUN 211
DTEVENT_MODULE_PLUGGED 211
DTEVENT_MODULE_UNPLUGGED 212
DTEVENT_OUT_BUFFER_SPACE_AVAILABLE 213
DTEVENT_OUTPUT_SCAN_BLOCK_UNDERRUN 214
DTEVENT_REMOTE_CHAN_CLOSED 214

DTEVENT_REMOTE_CHAN_OPENED 215
DTEVENT_SYNC_DATA_RECEIVED 215
DTSTATUS enumerated type
  in Visual Basic 178
  in Visual C++ 131

E

EDGE_TYPE enumerated type
  in Visual Basic 181
  in Visual C++ 134
enumerated types
  CLOCK_TYPE 126
  in Visual Basic 169
  in Visual C++ 123
error constants 122
  in Visual Basic 169
  in Visual C++ 121
error handling 34
ERROR_MODE_SELECT enumerated type
  in Visual Basic 181
  in Visual C++ 135
errors 107, 108
  DT_9841_MSG_UNKNOWN_TYPE 114
  DT_AD_VALUE_OUT_OF_RANGE 109
  DT_BOARD_ALREADY_OPEN 113
  DT_BOARD_NOT_RESPONDING 113
  DT_BOARD_WAS_SHUTDOWN 113
  DT_BUFFER_TOO_SMALL 108
  DT_CALLBACK_NOT_FOUND 108
  DT_CANT_FIND_BOARD 111
  DT_CANT_OPEN_DRIVER 111
  DT_CHAN_ALREADY_CLOSED 109
  DT_CHAN_ALREADY_OPEN 109
  DT_CHAN_DISCONNECTED 109
  DT_CHAN_OPENED_FOR_READ 110
  DT_CHAN_OPENED_FOR_WRITE 110
  DT_DRIVER_VERSION 111
  DT_DSP_VERSION 112
  DT_ERROR_FILE_READ 112
  DT_ERROR_INVALID_COFF 112
  DT_ERROR_MEMORY_MISMATCH 112
  DT_FAILURE 110
  DT_FILE_NOT_FOUND 111
  DT_FIRMWARE_VERSION 112
  DT_HOST_BUFFER_EMPTY 113
  DT_HOST_CHAN_ALREADY_CLOSED 113
  DT_HOST_CHAN_ALREADY_OPEN 113
  DT_HOST_CHAN_BUFFER_FULL 114
  DT_HOST_MSG_INVALID_LENGTH 114
  DT_HOST_RCV_BUFFER_FULL 114
  DT_HOST_SEND_BUFFER_FULL 114
  DT_INCOMPATIBLE_CHAN_TYPE 109
  DT_INVALID_ BOARD_HANDLE 110
  DT_INVALID_ BOARD_INDEX 110
  DT_INVALID_ BOARD_INFO 110
  DT_INVALID_ BOARD_NAME 110
  DT_INVALID_ PHBOARD 110
  DT_INVALID_ PHCHAN 108
  DT_INVALID_BUF_SIZE 110
  DT_INVALID_BUFFER_ADDRESS 112
  DT_INVALID_BUFFER_PTR 108
  DT_INVALID_BYTES_READ_PTR 108
  DT_INVALID_CALLBACK_PTR 108
  DT_INVALID_CHAN_HANDLE 108
  DT_INVALID_CHAN_NAME 108
  DT_INVALID_CHAN_TYPE 108
  DT_INVALID_DSP_ADDRESS_PTR 113
  DT_INVALID_DSP_BUFFER_SIZE_PTR 109
  DT_INVALID_FILENAME 110
  DT_INVALID_HOST_BUFFER_SIZE_PTR 109
  DT_INVALID_MSG_BUF_LEN 110
  DT_INVALID_MSG_PTR 110
  DT_INVALID_NUM_BYTES 108
  DT_INVALID_POINTER 108
  DT_INVALID_REGISTER 113
  DT_INVALID_WINDOW_HANDLE 110
  DT_NO_BOARD_FOR_INDEX 111
  DT_NO_CHAN_FOR_INDEX 111
  DT_NO_MEMORY 108
  DT_NO_MESSAGING 111
  DT_NO_PROGRAM_LOADED 112
  DT_NO_SYNC_DATA_RECEIVED 109
  DT_NOT_IMPLEMENTED 108
  DT_PROGRAM_TOO_LARGE 113
  DT_SUCCESS 108
  DT_TIMEOUT 108
  DT_TOO_BIG_FOR_BOARD_BUFFER 109
  DT_TOO_BIG_FOR_CHAN_BUFFER 109
  DT_USB_ERROR 112
  DT_VOLTAGE_OUT_OF_RANGE 109
  DT_WINDOW_NOT_FOUND 111
event-driven operations, how to perform 39
events
  DTEVENT_AD_OVERRUN 208
  DTEVENT_ASYNC_DATA_RECEIVED 208
  DTEVENT_CHAN_IN_BUFFER_FULL 209
  DTEVENT_DA_UNDERRUN 210
  DTEVENT_ERROR 210
  DTEVENT_INPUT_SCAN_BLOCK_OVERRUN 211
  DTEVENT_MODULE_PLUGGED 211

DTEVENT_MODULE_UNPLUGGED 212
DTEVENT_OUT_BUFFER_SPACE_AVAILABLE 213
DTEVENT_OUTPUT_SCAN_BLOCK_UNDERRUN 214
DTEVENT_REMOTE_CHAN_CLOSED 214
DTEVENT_REMOTE_CHAN_OPENED 215
DTEVENT_SYNC_DATA_RECEIVED 215

G

GATE_TYPE enumerated type
in Visual Basic 182
in Visual C++ 135
general-purpose constants, in Visual C++ 117

H

handling errors 34
help 8, 17

I

IEPE_CHAN_INFO data type, Visual Basic 203
IEPE_CHAN_INFO structure, Visual C++ 153
IEPE_SETUP data type, Visual Basic 203
IEPE_SETUP structure, Visual C++ 154
INPUT_SCAN_BLOCK structure, Visual C++ 155
INPUT_SCAN_RCD data type, Visual Basic 204
INPUT_SCAN_RCD structure, Visual C++ 156

M

message constants, in Visual C++ 122
MODEL_ID enumerated type
in Visual Basic 183
in Visual C++ 137
module name 23
MSG_INFO structure, Visual C++ 158

O

opening a communication channel 26
opening a module 23
OUTPUT_SCAN_BLOCK structure, Visual C++ 158
OUTPUT_SCAN_RCD data type, Visual Basic 205
OUTPUT_SCAN_RCD structure, Visual C++ 160

P

PMSG_INFO structure, Visual C++ 158
POLARITY enumerated type
in Visual Basic 185
in Visual C++ 138

R

reading data from a communication channel 27
reading from memory 28
reading from registers 28
register constants
in Visual Basic 166
in Visual C++ 119
registers
reading from 28
writing to 30
related documents 8
returning channel information 26
returning error messages 34
returning the module name 23
returning the number of bytes in a channel buffer 27
running a DSP program 25

S

structures
*PDATA_FILE_HDR 151
DATA_FILE_HDR 151
structures, Visual C++ 139

T

technical support 8
TRIGGER_SETUP data type, Visual Basic 206
TRIGGER_SETUP structure, Visual C++ 160
TRIGGER_TYPE enumerated type
in Visual C++ 139
troubleshooting checklist 17
typical communication operations, how to perform 37

W

writing data to a communication channel asynchronously 29
writing data to a communication channel synchronously 29
writing to memory 30
writing to registers 30